












Blockwoche: Web Programming Lab

Programm Blockwoche

| Montag  | Dienstag   | Mittwoch   | Donnerstag    | Freitag    |
|---|---|--|---|--|
| Architekturansätze von Web Anwendungen JavaScript Sprachkonzepte I | Client-Side- JavaScript I | Angular | Angular | Progressive Web Apps |
| JavaScript Sprachkonzepte II | Client-Side- JavaScript II | Angular | Server-Side- JavaScript REST | Authentication @ Web Apps |

Input

TypeScript



JavaScript «Issues»



PROGRAMMING PRO TIP

**CODE JAVASCRIPT UNDERWATER,
SO NOBODY COULD SEE YOU CRYING**

STARTED REFACTORING

COULDN'T STOP

memegenerator.net

JavaScript «Issues»

- Weak Typing bringt Vorteile, aber auch Herausforderungen
- JavaScript war ursprünglich für Client Side Event-Handling gedacht
(entwickelt 1995 von Netscape)
- Fehlende Sprachkonstrukte für “grössere” Anwendungen
(z.B. Interfaces)
- Prototype Inheritance gewöhnungsbedürftig

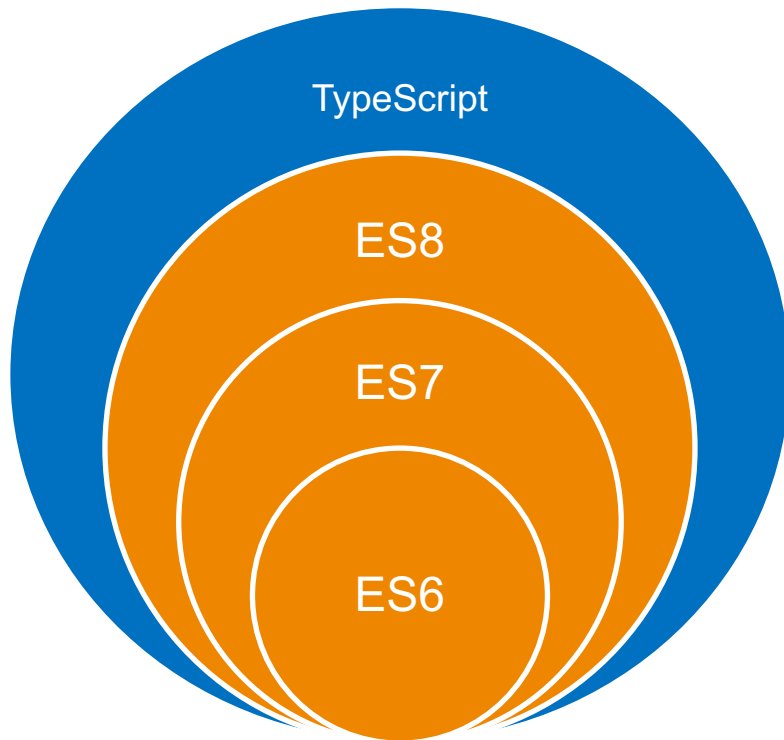
Sprachen, die nach JavaScript transpilieren

- TypeScript
- CoffeeScript
- Coco/LiveScript
- PureScript
- Elm
- Scala.js
- ClosureScript
- Dart/Dart2JS

Alternativen zu TypeScript & Co.

- ES.NEXT (mit Babel Compiler)
- Flow: Static Type Checker for JavaScript
- JSDoc: Type Safe JavaScript via Kommentare

TypeScript



TypeScript

- Superset von JavaScript
- Entwickelt von Microsoft
- Optionale statische Typisierung
- Basierend auf Klassen und Modulen
- Unterstützt Generics
- Wird zu JavaScript übersetzt
- Kompatibel mit ES8

Flexible Options

Any Browser

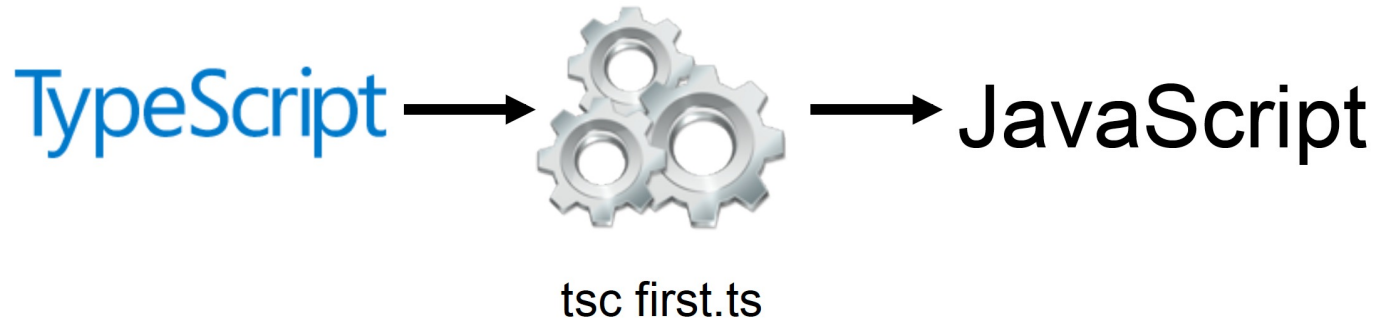
Any Host

Any OS

Open Source

Tool Support

TypeScript Transpiler



Let's dive into code....

Setup

- Editor wie z.B. Visual Studio Code oder IntelliJ
- NodeJS
- Node Package Manager (npm)
- Chrome
- git

→ git clone <https://github.com/web-programming-lab/typescript-jest-seed>

→ cd ./typescript-jest-seed && npm install

→ npm run start

→ npm run test

Types

- **Boolean, numbers & Strings**

```
let isDone: boolean = false;  
  
let goodNumber: number = 6;  
  
let color: string = "blue";
```


Types

- **Arrays**

```
let list: Array<number> = [1, 2, 3];
```

Types

- Enums

```
enum Color { Red, Green, Blue }  
  
let c: Color = Color.Green;
```

Types

- **Any**

```
let notSure: any = 4;
```

Interfaces

```
interface LabelledValue {  
    label: string;  
}  
  
function printLabel(labelledObj: LabelledValue) {  
    console.log(labelledObj.label);  
}  
  
let myObj = {size: 10, label: "Size 10 Object"};  
printLabel(myObj);
```

Read only properties

```
interface Point {  
    readonly x: number;  
    readonly y: number;  
}  
  
let p1: Point = { x: 10, y: 20 };  
p1.x = 5; // error!
```

Function Types

```
interface SearchFunc {  
    (source: string, subString: string): boolean;  
}  
  
let mySearch: SearchFunc;  
  
mySearch = function (source: string, subString: string) {  
    let result = source.search(subString);  
    return result > -1;  
}
```

Klassen Sichtbarkeiten

```
class BachelorModule {  
    private instructor: string;  
    private title: string;  
    private credits: number;  
  
    constructor(title: string) {  
        this.title = title;  
    }  
}  
  
let webApp = new BachelorModule("WebApp");  
console.log(webApp.title); // Error
```

Getter / Setter (auch ab ES6 möglich)

```
class BachelorModule {  
    private _title: string;  
  
    constructor(title: string) {  
        this._title = title;  
    }  
  
    get title(): string {  
        return this._title;  
    }  
}  
  
let webApp = new BachelorModule("WebApp");  
console.log(webApp.title); // Ok
```


Vererbung (auch ab ES6 möglich)

```
class BDAModule extends BachelorModule {  
    private _expertName: string;  
  
    set expertName(name: string) {  
        this._expertName = name;  
    }  
}  
  
let bda = new BDAModule("SomeTitle");
```

Generics

```
function identity(arg: number): number {  
    return arg;  
}
```

```
function identity(arg: any): any {  
    return arg;  
}
```

Generics

```
function identity<T>(arg: T): T {  
    return arg;  
}  
  
let out = identity<string>("Hello");  
console.log(out);
```

Übung 1: Interface

```
class App {  
    createUser(name: string): User {  
        return {  
            id: 123,  
            name: name,  
            settings: {  
                settingA: 123,  
                settingB: 456,  
            }  
        }  
    }  
}
```

- Erstellt für «User» und «Settings» Interfaces
- Branch in GitHub Repo: interface

Übung 2: Generics

```
interface KeyPair {  
    key: string;  
    value: string;  
}  
  
let kv1: KeyPair = { key:"key", value:"value" };  
let kv2: KeyPair = { key:1, value:"Steve" };  
let kv3: KeyPair = { key:1, value:12345 };
```

- Korrigiere die Kompilierungsfehler mit Generics
- Branch in GitHub Repo: generics

Übung 3: Two-for-one

- Erhalte einen Namen als Input und gib folgendes auf der Konsole aus (X ist der Name):

«One for X, one for me.»

- Wenn der Name fehlt, gib folgendes aus:

«One for you, one for me.»

- Branch in GitHub Repo: two-for-one

Modulsystem: Ziele

- Abhängigkeiten zwischen den einzelnen Komponenten möglichst gering halten.
- Standardisierter Einsatz von Schnittstellen zwischen den Modulen.
- Förderung einer besseren Wartbarkeit und Wiederverwendbarkeit des Codes.

Modulsysteme in JavaScript

«Modules as a concept in JavaScript have a long and complicated history that makes any single definition or description difficult.»

typescriptlang.org

<https://www.typescriptlang.org/docs/handbook/2/modules.html>

Modulsysteme in JavaScript & TypeScript

- **Vor ES6:** Modul Emulationen, z.B. via
 - CommonJS
 - requireJS (*implementiert die “Asynchronous Module Definition” AMD API*)
- **Ab ES6:**
 - Mit ES6 enthält JavaScript ein eigenes Modulsystem
 - SystemJS: Module Loader für ES6 Module und Backward Compatibility
- **TypeScript:**
 - TypeScript teilt das Konzept der Modularisierung von ES6
 - TypeScript unterstützt die Modulsysteme ES6, CommonJS, SystemJS und AMD

Module in TypeScript

- Alle definierten Variablen, Funktionen, Klassen etc. sind nur im Modul sichtbar, ausser sie werden explizit **exportiert**.
- Um eine Variable, Funktion, Klasse etc. von einem anderen Modul zu nutzen, muss sie entsprechend **importiert** werden.
- Ein ts-File mit einem top-level “export” oder “import” wird als Modul angesehen.
- Ein ts-File ohne top-level “export” oder “import” wird als Script angesehen und ist daher global verfügbar.

Module in TypeScript

StringValidator.ts

```
export interface StringValidator {  
    isAcceptable(s: string): boolean;  
}
```

ZipCodeValidator.ts

```
import { StringValidator } from "../StringValidator";  
  
export const numberRegexp = /^[0-9]+$/;  
  
export class ZipCodeValidator implements StringValidator {  
    isAcceptable(s: string) {  
        return s.length === 5 && numberRegexp.test(s);  
    }  
}
```

Ambient Declarations

- Dank Ambient Declarations können JavaScript Libraries “typisiert” in TypeScript eingesetzt werden.
- Type Definition Files: File zum Mapping von Typ-Definitionen einer JavaScript Library (z.B. jQuery)
- DefinitelyTyped: Populäres GitHub Repository für Typ-Definitionen

```
/// <reference path="jquery.d.ts" />
import $ from "jquery";

var data = "Hello John";

$("#div").text(data);
```

Sourcemaps

- File zum Mapping zwischen originalem und generiertem Code
- Ermöglicht Debugging originalem Code (z.B. TypeScript Code)
- Beispiel:

```
{"version":3,"file":"app.js","sourceRoot":"","sources":["../src/app.ts"],  
,"names":[],"mappings":";;AAAA,MAAqB,MAAM;IAA3B;QACU,....."
```

Zusammenfassung

- TypeScript ist ein Superset von JavaScript
- TypeScript enthält eine optionale statische Typisierung
- TypeScript wird zu JavaScript übersetzt
- TypeScript ist kompatibel mit ES8

Übungsfragen

1. Warum wird TypeScript als Alternative zu JavaScript eingesetzt?
2. Was macht der TypeScript Compiler (tsc)?
3. Können Browser TypeScript native interpretieren?
4. Wie ist in TypeScript ein Modul definiert?
5. Was sind Ambient Declarations in TypeScript?
6. Was sind Sourcemaps und wozu werden sie benötigt?

Übung 4: Charles

Charles gibt immer dieselben Antworten. Erhalte eine Message an Charles als Input und gib seine Antwort auf der Konsole aus:

- Charles antwortet 'Sure.' wenn Du ihm eine Frage stellst.
- Er antwortet 'Whoa, chill out!' wenn Du ihn anschreist.
- Er sagt 'Fine. Be that way!' wenn Du ihn in deiner Nachricht anschweigst.
- Er antwortet 'Whatever.' zu allem anderen.
- Branch in GitHub Repo: charles

Übung 5: Classlist

Erhalte Informationen zu Studenten und ihren jeweiligen Klassen und erstelle daraus eine Klassenliste. Implementiere folgende Funktionen:

- Student hinzufügen (Name und Klasse)
- Liste aller Studenten einer Klasse ausgeben
- Sortierte Liste aller Studenten in allen Klassen ausgeben. Klassen starten mit 1, 2, 3 etc., die Studenten sind alphabetisch nach Name sortiert.
- Branch in GitHub Repo: classlist

Zusatz-Übung: Math-Bot

Erstelle einen Chat-Bot in TypeScript, der Mathematik-Aufgaben aus Textanfragen lösen kann:

- Beispiel: «Was ist 7 minus 5?» \rightarrow 2
- Unterstützte Operationen: Addition, Subtraktion, Multiplikation, Division
- Der Chat-Bot weist Anfragen zurück, die er nicht beantworten kann
- Branch in GitHub Repo: mathbot

Input CSS



CSS

- CSS (*Cascading Style Sheets*) ist eine Formatierungssprache für elektronische Dokumente.
- CSS wurde entworfen, um Darstellungsvorgaben vom Inhalt zu trennen.
- Regelbasierter Mechanismus, welcher Style auf einem Element schlussendlich angewendet wird (Cascade).

CSS Style Basics

Element Selector

propertyname:value

Id Selector

Class Selector

```
body {  
    background-color: #cccc99;  
    padding: 10px;  
}  
  
#menu {  
    background-color: #ffff00;  
}  
  
.bookTitle {  
    font-style: italic;  
}
```

CSS Style Basics

Descendant selector



Child selector



Attribute Selector



Pseudo Class



```
div p {  
    background-color: #dddada;  
}  
  
div > p {  
    background-color: #dddada;  
}  
  
img[alt = spacer] {  
    padding: 0px;  
}  
  
a:hover {  
    color: #dddddd;  
}
```

CSS Cascade

```
<p class="better">This is a paragraph.</p>  
<p class="better" id="winning">One selector to rule them all!</p>
```

```
#winning {  
    background-color: red;  
    border: 1px solid black;  
}  
.better {  
    background-color: gray;  
    border: none !important;  
}  
p {  
    background-color: blue;  
    color: white;  
    padding: 5px;  
}
```

CSS Cascade

```
<p class="better">This is a paragraph.</p>  
<p class="better" id="winning">One selector to rule them all!</p>
```

```
#winning {  
  background-color: red;  
  border: 1px solid black;  
}  
.better {  
  background-color: gray;  
  border: none !important;  
}  
p {  
  background-color: blue;  
  color: white;  
  padding: 5px;  
}
```

This is a paragraph.

One selector to rule them all!

Übung: CSS Selectors

- CSS Diner WebApp
- <http://flukeout.github.io/>

Integration von CSS in HTML

Direkt via Style-Attribut

```
<div style="color: yellow;"></div>
```

Einbinden von externen Stylesheets:

```
<!doctype html>  
<html lang="de">  
<head>  
<link rel="stylesheet" href="stylesheet.css">  
...
```

Angabe von CSS-Klasse im HTML Element:

```
<div class="hero"></div>
```

CSS Media Queries

```
/* Set the background color of body to tan */
body {
    background-color: tan;
}

/* On screens that are 992px or less, set the background color to blue */
@media screen and (max-width: 992px) {
    body {
        background-color: blue;
    }
}

/* On screens that are 600px or less, set the background color to olive */
@media screen and (max-width: 600px) {
    body {
        background-color: olive;
    }
}
```

CSS Flexbox

- Responsive Layoutmodell zur Platzverteilung zwischen Elementen
- Flexbox lernen mit dem Flexbox Froggy: <https://flexboxfroggy.com>

```
#pond {  
  display: flex;  
  justify-content: flex-end;  
}
```



Übung CSS Flexbox

- Flexbox lernen mit dem Flexbox Froggy
- <https://flexboxfroggy.com>

CSS Präprozessoren

- Erweiterungen von CSS, welche zur Ausführung nach CSS umgewandelt werden
- Erhöht die Wartbarkeit sowie Wiederverwendbarkeit des CSS Codes
- Beliebte CSS Präprozessoren:
 - Sass (Sass oder SCSS)
 - LESS
 - Stylus

CSS Präprozessoren: Variablen in SCSS

```
// variable
$color-green: #91ea42;

.text-green {
  color: $color-green;
}

.button-green {
  background: $color-green;
}
```

CSS Präprozessoren: Imports in SCSS

```
// shared modules
@import 'modules/_module-1';

// view specific css
@import 'views/_home';
@import 'views/_features';
```


CSS Präprozessoren: Mixins in SCSS

```
// mixin
@mixin transform($property: scale3d(2, 1.5, 0.5)) {
  -webkit-transform: $property;
  -ms-transform: $property;
  transform: $property;
}

// include mixin
div {
  @include transform();
}
```

CSS Präprozessoren: Nestings in SCSS

SCSS

```
// navigation
nav {
  ul {
    list-style: none;
  }
  li {
    display: inline-block;
  }
  a {
    display: block;
    text-decoration: none;
    padding: 0.5rem;
    :hover {
      text-decoration: underline;
    }
  }
}
```

CSS

```
nav ul {
  list-style: none;
}
nav li {
  display: inline-block;
}
nav a {
  display: block;
  text-decoration: none;
  padding: 0.5rem;
}
nav a :hover {
  text-decoration: underline;
}
```

Zusammenfassung

- CSS (*Cascading Style Sheets*) ist eine Formatierungssprache für elektronische Dokumente.
- Der regelbasierte Mechanismus von CSS entscheidet, welcher Style auf einem Element angewendet wird (Cascade).
- CSS Flexbox und Media Queries werden zur Erstellung von Responsive Layouts eingesetzt.
- CSS Präprozessoren wie Sass oder LESS erhöhen die Warbarkeit und Wiederverwendbarkeit von CSS.