Allgemeines

- Syntaktische Obermenge von JavaScript
- Fügt statische Typisierung hinzu
- Ermöglicht Angabe der Datentypen
- Typüberprüfung erfolgt vorab während Kompilierung
- JavaScript:
  - Sprache mit loser Typisierung
  - Typ von Variablen und Funktionsparametern wird durch ihre Verwendung festgelegt
- TypeScript Compiler wandet TypeScript Code in JavaScript Code um
- Installation des Compilers mittels node.js Package Manager (npm)
- Ausführung des Compilers mittels npx
- TypeScript-Compiler wird durch Datei tsconfig.json konfiguriert
- Initiierung durch das Kommando npx tsc -init
- Festlegung z.B. von Eingabe- und Ausgabeverzeichnissen

```
{
    "include": ["src"],
    "compilerOptions": {
        "outDir": "./build"
    }
}
```

Typen

- Typen in TypeScript können explizit oder implizit festgelegt werden
- Explizite Festlegung: Variablenname gefolgt von Doppelpunkt und Typ
- Typen: **number, string, boolean**
- Bei impliziter Typ-Festlegung schließt TypeScript den Typ auf Basis der ersten Zuweisung
- In beiden Fällen ist der Typ der Variablen hinterher String (Zeichenkette)

```
// explizite Typ-Festlegung
let vorname: string = "Olaf";

// implizite Typ-Festlegung
let vorname2 = "Friedrich";
```

- Spezielle Typen: **any** und **unknown**
- Wenn TypeScript auf einen Typ nicht schließen kann, wird die Typisierung hierfür außer Kraft gesetzt (Typ: **any**)
- Verhindert keine falsche Nutzung
- Besser: Typ explizit auf unbekannt (**unknown**) setzen und vor der Verwendung ggf. auf korrekten Typ casten

```
let ergebnis: any = 33;
ergebnis = "Christian"; // kein Fehler!
if (ergebnis == false)  // kein Fehler!
    ergebnis = true;

let resultat: unknown = 11;
```

- Fehler bei Typ-Wechsel: Fehlermeldung, wenn Typ eines zugewiesenen Wertes und zuvor (explizit oder implizit) festgelegter Typ nicht übereinstimmen!

```
// explizite Typ-Festlegung
let vorname: string = "Olaf";
vorname = 3.14;

// implizite Typ-Festlegung
let vorname2 = "Friedrich";
vorname2 = true;
```

Arrays

- Zusätzliche eckige Klammern nach dem Typ
- Alle Array-Elemente haben den gleichen Typ!
- Auch bei Arrays kann die Typ-Zuweisung implizit erfolgen
- Arrays, deren Inhalte nicht verändert werden können, verwenden zusätzlich das Attribut **readonly**

```
const namen: string[] = [];
// JS: const name = [];
name[0] = "Robert";
name[1] = 42;

const vornamen: readonly string[] = [];
vornamen[0] = "Annalena";
```

Tupel

- Typisiertes Array mit einer vordefinierten Länge und festgelegten Typen
- Für jeden Index wird der Typ vorab explizit festgelegt
- Für höhere Indexwerte ist der Typ immer **any**
- Höhere Indexwerte ausschließen mit **readonly**

```
let meinTupel: [string, number, boolean];
meinTupel = ['Hubertus', 51, true];
meinTupel[2] = 12; // Fehler!
meinTupel[4] = "Boris"; // kein Fehler!

let meinAnderesTupel: readonly [string, number];
meinAnderesTupel = ['Nancy', 53];
meinAnderesTupel[2] = true; // Fehler!
```