

The background of the slide features a close-up photograph of a squirrel's head and shoulders. The squirrel has brown fur and is looking slightly to the right. It appears to be eating some small, round, brown objects, likely nuts, which are scattered on the surface in front of it.

NILS HARTMANN

<https://nilshartmann.net>
<https://reactbuch.de>

Let's type!

TypeScript

Eine interaktive Einführung

Slides: <https://nils.buzz/devtalk-tk-typescript>

NILS HARTMANN

nils@nilshartmann.net

Freiberuflicher Entwickler, Coach und Trainer

Java
JavaScript, TypeScript
React
GraphQL

**Trainings, Workshops und
Beratung**



<https://reactbuch.de>

HTTPS://NILSHARTMANN.NET

"TypeScript is probably the most important language right now (...)

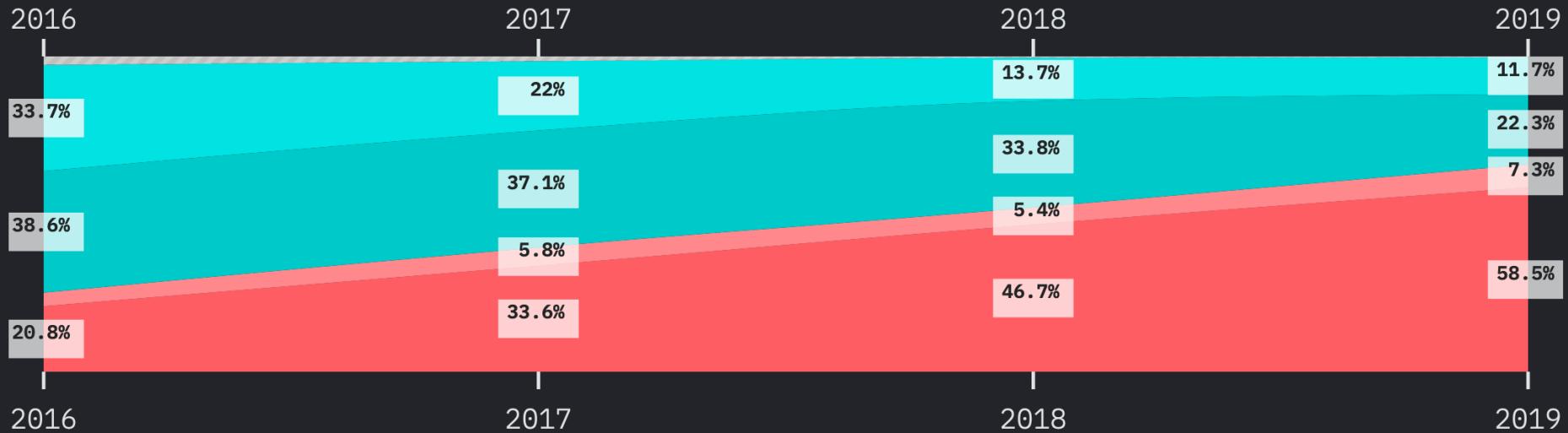
TypeScript makes JavaScript twice as good, and that's a conservative estimate (...)

In terms of impact, TypeScript is the most important thing right now possibly."

-- Rod Johnson ("Erfinder" von Spring), Januar 2018
[\(https://thenewstack.io/spring-rod-johnson-enterprise-java/\)](https://thenewstack.io/spring-rod-johnson-enterprise-java/)

TypeScript

Respondent's experience with TypeScript.



- I've USED it before, and WOULD use it again
- I've HEARD of it, and WOULD like to learn it
- I've never heard of it

- I've USED it before, and would NOT use it again
- I've HEARD of it, and am NOT interested

<https://2019.stateofjs.com/javascript-flavors/typescript/>

TypeScript: Obermenge von JavaScript

- Jeder gültige JavaScript Code ist gültiger TypeScript Code (theoretisch...)
- Ergänzt JS um Typ-System, Sichtbarkeiten, Enums und Dekoratoren
- Compiler erzeugt aus TypeScript-Code JavaScript (ES3, ES5, ES6)-Code
- Entwickelt von Microsoft
 - <http://www.typescriptlang.org/>
- Sehr guter IDE Support
 - IntelliJ IDEA, Visual Studio Code

```
03_x_datasource.ts — typescript-intro

15 |     url: string | null;
16 |     authentication: UsernamePasswordAuthentication | SimpleAuthentication;
17 |
18 |
19 | const baseConfig: DataSourceConfig = {
20 |     url: "http://www.voxxedvienna.at",
21 |     authentication: "nils@secret"
22 | };
23 |
24 | const config: DataSourceConfig = baseConfig;
25 |
26 | createDataSource(config);
27 |
28 | interface DataSource {}
29 |
30 | function createDataSource(config: DataSourceConfig): DataSource {
31 |     if (config.url !== null) {
32 |         const myUrl = config.url.toLocaleLowerCase();
33 |         authentication
34 |         url
35 |         if (isBasicAuth(config.authentication)) {
36 |             config.authentication.toLocaleLowerCase();
37 |         } else {
38 |             config.authentication.password;
39 |         }
40 |
41 |         // return a datasource ...
42 |         return {};
43 |     }
44 |
45 | }
```

Praktische Einführung!

<https://github.com/nilshartmann/typescript-intro/>

Syntax

TYPE ANNOTATIONS

Typen verwenden

Variablen

```
let foo: string; // built-in types, for example: string, number, boolean
```

TYPE ANNOTATIONS

Typen verwenden

Variablen

```
let foo: string; // built-in types, for example: string, number, boolean
```

Funktionen

```
function sayIt(what: string) {  
    return `Saying: ${what}`;  
}
```

TYPE ANNOTATIONS

Using Types

Variables

```
let foo: string; // built-in types, for example: string, number, boolean
```

Functions

```
function sayIt(what: string) {  
    return `Saying: ${what}`;  
}
```

Typ Angaben sind optional, Typ wird von TS abgeleitet

```
let result = 7; // abgeleiteter Typ: number  
result = sayIt('Lars') // Fehler! (abgeleiteter Typ von sayIt: string)
```

TYPE ANNOTATIONS

any Type

```
let foo: any; // erlaubt alle Typen, kein Typ-Checking findet mehr statt  
  
foo = "Klaus"; // OK  
foo = 7; // OK  
foo = null; // OK
```

TypeScript weist any implizit immer dann zu, wenn TS keinen Typ bestimmen kann:

```
function sayWhat(s) {  
    // s ist any  
}
```

Im "strict mode" weist TypeScript nie "any" zu, stattdessen gibt es einen Fehler (man kann aber selber any verwenden)

TYPE ANNOTATIONS

unknown Type

```
let foo: unknown; // erlaubt alle Typen, zwingt aber zur Überprüfung vor Nutzung
```

```
foo = "Klaus"; // OK
foo = 7; // OK
foo = null; // OK
```

```
foo.toUpperCase(); // ERROR
if (typeof foo === "string") {
  foo.toUpperCase(); // OK
}
```

UNION TYPES

Ein *Union Type* zeigt an, dass ein Wert *verschiedene Typen* haben

```
let foo: string | number;  
  
foo = 7; // OK  
foo = "Seven"; // auch OK  
  
foo = false; // Fehler
```

NULL UND UNDEFINED

null und undefined sind eigene Typen in TypeScript

- Typen sind nicht nullable und können nicht undefined sein (mit "strictNullChecks")

```
let a: string = "Klaus";
a = null; // Error
```

Mit Union type können wir null zulassen:

```
let a: string | null = "Klaus;
a = null; // OK
```

Gleiches gilt für undefined:

```
let a: string | undefined;
```

Empfehlung: bei neuen Projekten "strictNullChecks" einschalten!

STRING LITERAL TYPE

String Literal Type

- Mit dem "String Literal Type" kann definiert werden, welche Werte ein String annehmen kann

```
type Language = "Java" | "TypeScript"; // Java oder TypeScript
```

```
const java:Language = "Java"; // OK
```

```
const cpp:Language = "C++"; // FEHLER
```

EIGENE TYPEN

Eigene Typen – Interfaces definieren "Struktur" eines Objektes

```
interface Person {          // alternativ: type
    firstName: string,
    lastName: string|null,   // nullable Type ("ein String oder null")
    age?: number             // optional type (darf undefined sein)
}
```

EIGENE TYPEN

Eigene Typen – Interfaces definieren "Struktur" eines Objektes

```
interface Person {          // alternativ: type
    firstName: string,
    lastName: string|null,   // nullable Type ("ein String oder null")
    age?: number             // optional type (darf undefined sein)
}

function sayHello(p: Person) {
    console.log(`Hello, ${p.lastName}`);
    p.lastName.toUpperCase(); // Fehler: "Object is possibly null"
}

sayHello({firstName: 'Klaus', lastName: null}); // OK
sayHello({firstName: 'Klaus', lastName: 777}); // Fehler: lastName not a string
sayHello({firstName: 'Klaus', lastName: 'Mueller', age: 32}); // OK
```

STRUKTURELLE IDENTITÄT

TypeScript arbeitet mit "Struktureller Identität" (*structural identity*)

```
interface Person {  
    name: string  
}
```

```
interface Animal {  
    name: string  
}
```

```
// Eine Person erzeugen...  
const p:Person = { name: 'Klaus' };
```

```
// ...person einem Animal zuweisen 🤔  
const a:Animal = p; // OK, da Person and Animal dieselbe Struktur haben  
                    // (wäre in Java/C# nicht erlaubt)
```

Class Syntax wie in ES6, aber mit Sichtbarkeiten

```
class Person {  
    private name: string  
  
    constructor(name: string) {  
        this.name = name;  
    }  
  
    const p = new Person("Klaus");  
    console.log(p.name); // FEHLER: "name" nicht sichtbar
```

Generics

```
interface Person { name: string };
interface Movie { title: string };

let persons:Array<Person> = [];
let movies:Array<Movie> = [];

persons.push({name: 'Klaus'});           // OK
movies.push({title: 'Batman'});          // OK
persons.push({title: 'Casablanca'}) // Fehler (Property 'title' not in Person)
```

TYPE CHECKING JAVASCRIPT CODE

Type Checking kann auch für JS Dateien eingeschaltet werden!

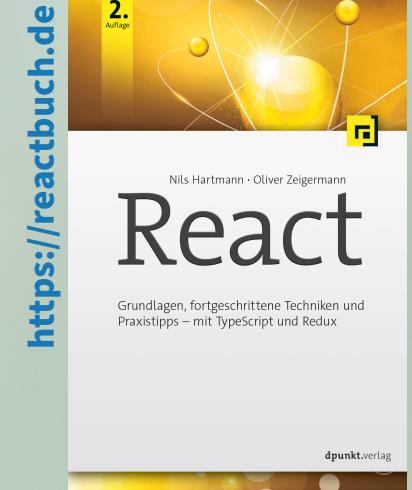
- Mit der *ts-check* Direktive am Anfang einer Datei
- "Typ-Informationen" können über JSDoc hinzugefügt werden

```
// @ts-check

/**
 * @param {string} name The name
 * @param {number} age The age
 */
function newPerson(name, age) {
  name.toLowerCase(); // OK
  age.toLowerCase(); // ERROR Property 'toLowerCase' does not exist on type 'number'
}
```

NILS HARTMANN

<https://nilshartmann.net>



vielen Dank!

Slides: <https://nils.buzz/devtalk-tk-typescript>

Source Code: <https://github.com/nilshartmann/typescript-intro>

Fragen & Kontakt: nils@nilshartmann.net

@NILSHARTMANN