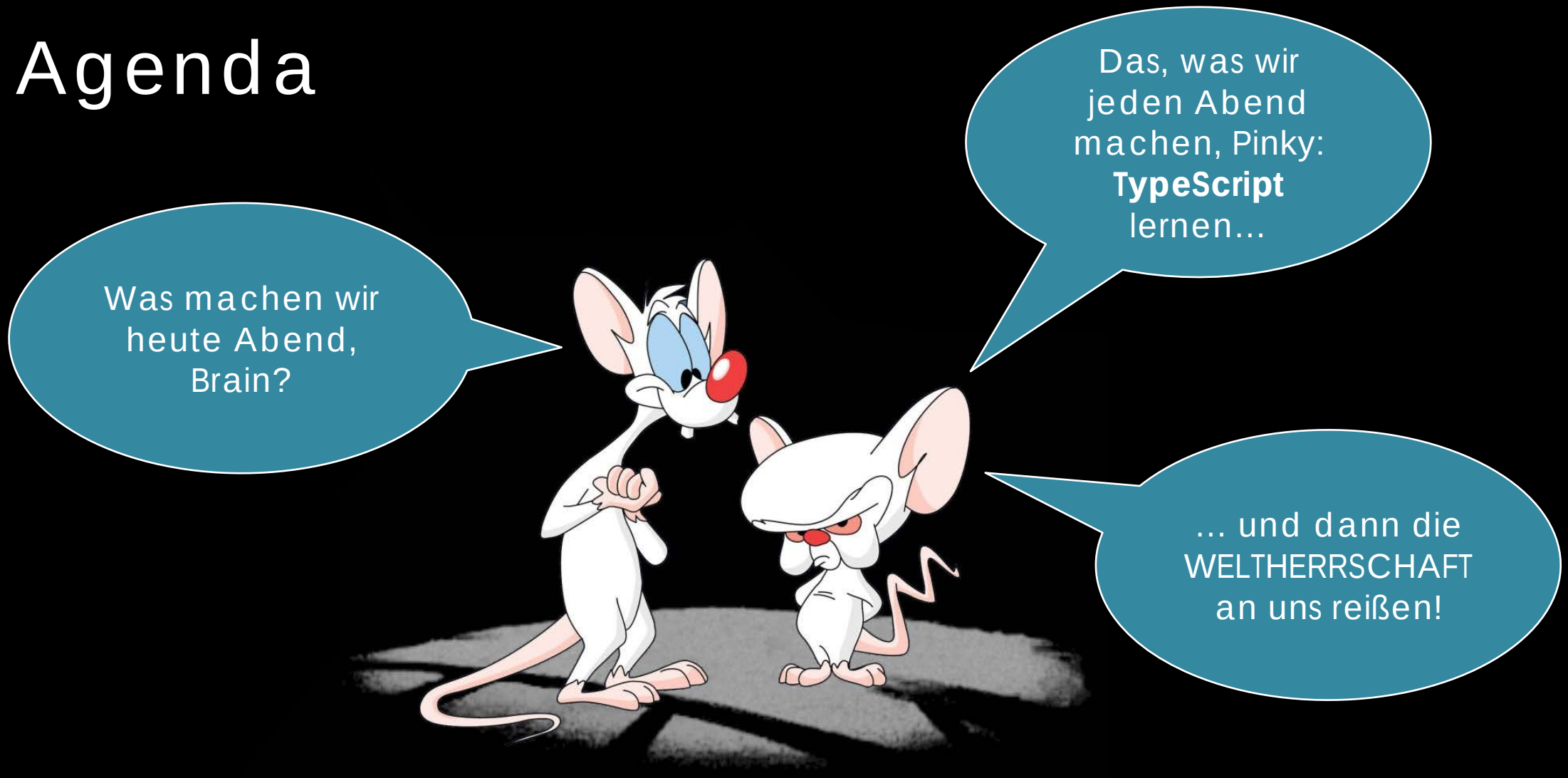
TYPESCRIPT

Für JS-Entwickler

Agenda


Was machen wir heute Abend, Brain?
Das, was wir jeden Abend machen, Pinky: **TypeScript** lernen...
... und dann die WELTHERRSCHAFT an uns reißen!

WTF, JavaScript?!

- Ein kleines Rätsel zum Warmwerden:

```
function add(a, b) {
    return a + b;
}
```

- Was ergibt...?

```
add(1, 3);                  // 4

add(1);                     // NaN

add(1, "1");                // "11"

add(1, {});                 // "1[object Object]"

add(1, []);                 // "1"
```

Was wollen wir eigentlich?

- **Eine Funktion, die zwei Zahlen erwartet und eine Zahl zurückgibt**
- Das ist TypeScript! – *„TypeScript extends JavaScript by adding types."*

```
function add(a: number, b: number): number
{
    return a + b;
}
```

- Resultat:

```
add(1);
~~~~~~          // Fehler: 2 Argumente wurden erwartet, empfangen wurden aber 1.

add(1, "1");
       ~~~      // Fehler: Das Argument vom Typ ""1"" kann dem Parameter
                   vom Typ "number" nicht zugewiesen werden.

                // ... analoge Fehler für die anderen Aufrufe
```

TypeScript schützt uns vor unseren eigenen Fehlern

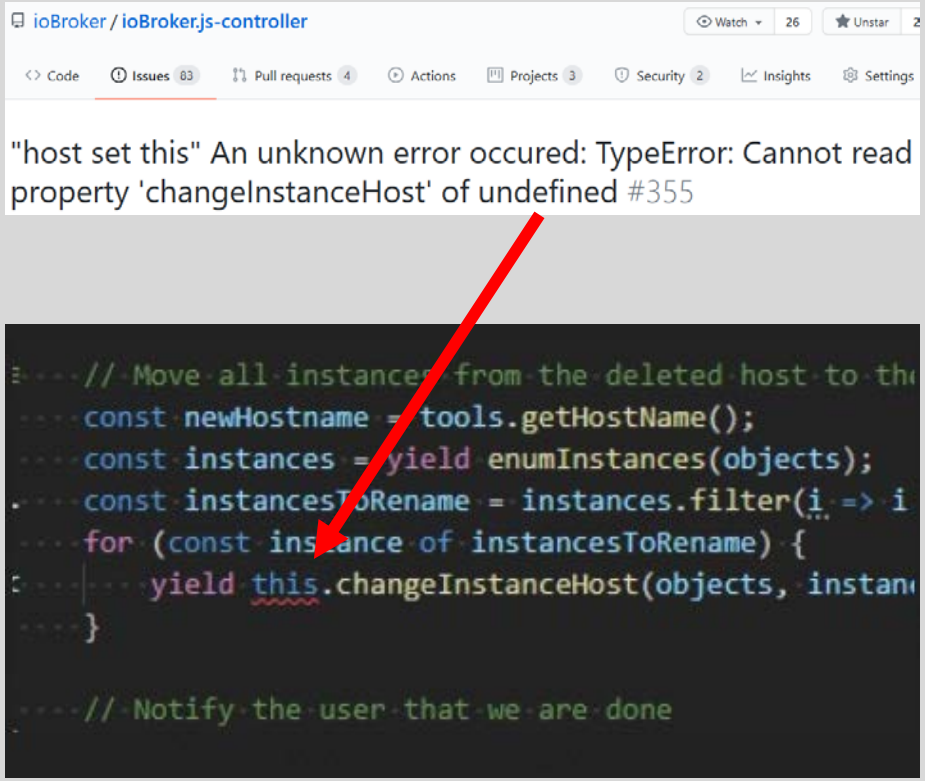
- Was kommt hier raus?

```
function addRounded(a, b) {
    return a.toFixed(3) + b.toFixed(3);
}

addRounded(1.123456, 2.234567);
```

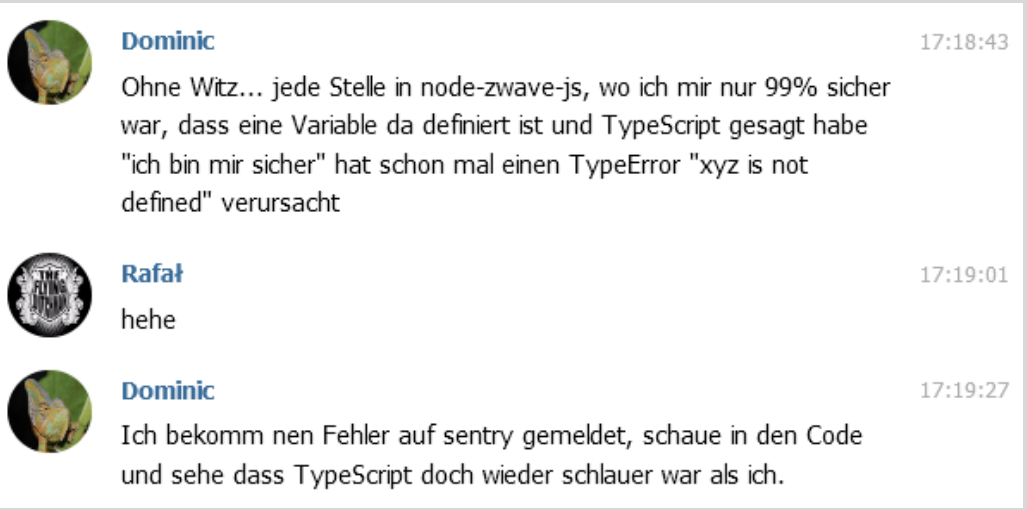
- Versuch 2 – diesmal mit TypeScript

```
function addRounded(a: number, b: number): number {
    return a.toFixed(3) + b.toFixed(3);
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    // Fehler Der Typ "string" kann dem Typ "number" nicht zugewiesen werden.
}
```

...ein paar Beispiele aus der Praxis



...selbst, wenn man denkt,
man weiß es besser...



TypeScript hilft beim Refactoring

```
function blabla(stuff) {
    console.log(obj.stuff.toFixed(3));
}

blabla({
    stuff: 1.2345
});

// 3000 Zeilen Code

blabla({
    stuff: 2.4567
});
```

```
function blabla(obj: {stuff: string}) {
    console.log(obj.stuff.toLowerCase());
}

blabla({
    stuff: "1.2345"
});

// 3000 Zeilen Code

blabla({
    stuff: 2.4567
    ~~~~~ // Fehler: Der Typ "number" kann dem Typ
          // "string" nicht zugewiesen werden
});
```

TypeScript schützt vor Laufzeitfehlern

- JavaScript:

```
function getValueOrNull() {
    return Math.random() > 0.5 ? 1 : null;
}

function doSomething() {
    const val = getValueOrNull();
    val.toString();
}

TypeError: Cannot read property 'toString' of null
```

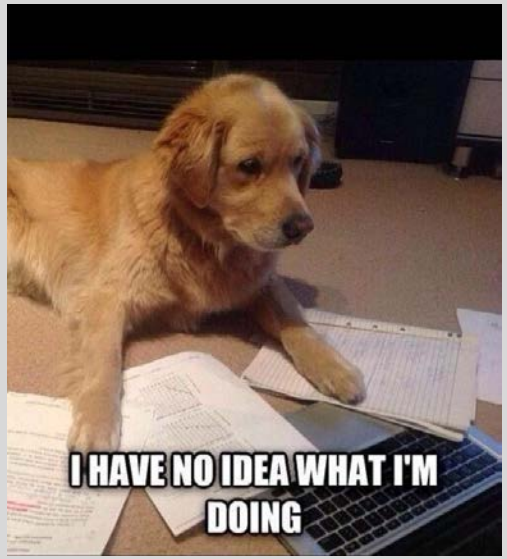
- TypeScript:
  (mit strictNullChecks)

```
function getValueOrNull() {
    return Math.random() > 0.5 ? 1 : null;
}

function doSomething() {
    const val = getValueOrNull();
    val.toString();
    ~~~ // Fehler: Objekt ist möglicherweise null
}
```

TypeScript hilft beim Nutzen von externen Libraries (1)

- JavaScript:

```
const axios = require("axios");

axios.???
```



- TypeScript:

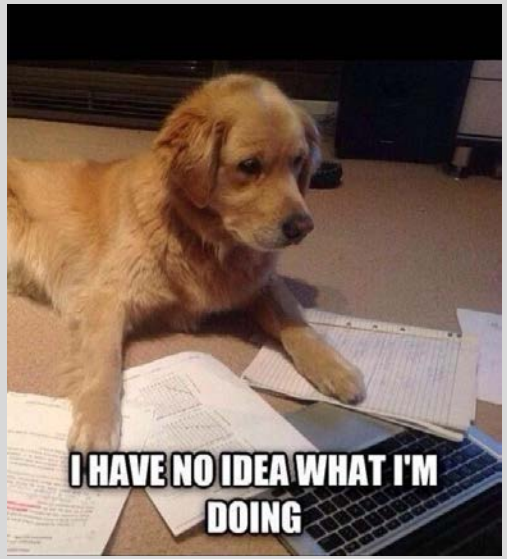
```
import axios from "axios";

axios.get(
```

```
get(url: string, config?: AxiosRequestConfig |
undefined): Promise<AxiosResponse<any>>
```

TypeScript hilft beim Nutzen von externen Libraries (2)

- JavaScript:

```
const axios = require("axios");

axios.get(123); // ok?
```



- TypeScript:

```
import axios from "axios";

axios.get(123);
          ~~~ // Fehler: Das Argument vom Typ "number"
              // kann dem Parameter vom Typ "string"
              // nicht zugewiesen werden.
```

Ein paar Grundlagen (1)

Einfache Typen

```
type Primitive =
    | number
    | string
    | boolean
    | null
    | undefined
;
```

Komplexere Objekte

```
interface MyObjectType {
    property1: number;
    property2: string;
    subProperty: {
        sub1: boolean;
    }
}
```

Komplexe / zusammengesetzte Typen

```
type NumberOrString = number | string;
type ArrayOfNumber = number[];
type ObjectWithProperty = { property: string };
type ArrayOfPrimitives = Primitive[];
```

Objekttypen

```
type SomeObject = Record<string, any>;
type SomeObjectFullOfNumbers = Record<string, number>;

// Bitte nicht (äquivalent zu irgendwas, das nicht Null ist):
function takesObject(obj: object)
function takesObject(obj: {})
```

Tupel

```
type TwoNumbersAndAString = [number, number, string];
type NamedTuple = [x: number, y: number];
```

Ein paar Grundlagen (2)

Sichere Verwendung externer Daten mit `unknown`

```
const thisCouldBeEverything: unknown = 1;

if (typeof thisCouldBeEverything === "number") {
    const addOne = thisCouldBeEverything + 1;
} else {
    const nope = thisCouldBeEverything.toString();
                 ~~~~~~~~~~~~~~~~~~~~~~
    // Fehler: Das Objekt ist vom Typ "Unbekannt"
}
```

Literal-Typen

```
type OneOrTwoOrEmptyString = 1 | 2 | "";
function acceptsTrue(val: true) {}
```

Typprüfung ausschalten mit `any`

```
const iDoNotCareWhatThisIs: any = 1;

function acceptsEverything(
    something: any
): any {
    return something;
}
```

Ein paar Grundlagen (3)

Funktionstypen

```
function returnsNothing(): void

async function resolvesToString(): Promise<string>

function takesCallback(
    cb: (arg1: string) => void
)

function neverExits(): never {
    process.exit(0);
}
```

Finger weg von:

```
type Nope =
    | Number
    | Boolean
    | String
    | Symbol
    | Function
    | Object
    | object
    | {}
;
```

Okay, let‘s get started

- TypeScript im Repo installieren (lokale Installation empfohlen) und initialisieren

```
$ npm i -D typescript
+ typescript@4.0.2
added 1 package from 1 contributor and audited 1 package in 0.931s
found 0 vulnerabilities

$ npx tsc --init
message TS6071: Successfully created a tsconfig.json file.
```

- TypeScript konfigurieren → Live-Demo
  - tsconfig.json
  - package.json - Skripte