Exploring Type Annotations and Type Inference in TypeScript

Chris B. Behrens
Senior Software Architect

@chrisbbehrens

"When" Is the Point

```
int x = 5;
x = “Chris B. Behrens”;
```

Another Example

```
int age;
var formValue = request[“age”];
age = int.Parse(formValue);
```

Real Type Coercion

```
int x = 10;
int xSquared = (int)Math.Pow(x, 2);
Console.WriteLine(xSquared);
```

Integer Is Fine

```
100 == 100.0
```

How type inference works in strongly-typed languages

JavaScript Typing

```
var x = 5;
x = “Chris B. Behrens”;

dynamic x = 5;
x = "Chris B. Behrens";

var x = 5;
console.log(typeof(x));
x = "Chris B. Behrens"
console.log(typeof(x));
```

Is the ability to change the type at will a good thing or a bad thing?

Type coercion happens more often due to an error in coding than by design with loose typing.

Broken Windows

Data annotations for Python

Dynamic typing can be great...

For senior developers

Small crimes lead to larger crimes

Strong typing forces you to think

The TypeScript "Types"

The Three-ish TypeScript Primitives

String

Number

Boolean

Booleans in JavaScript

"In JavaScript, a truthy value is a value that is considered true when encountered in a Boolean context. All values are truthy unless they are defined as falsy. That is, all values are truthy except false, 0, -0, 0n, "", null, undefined, NaN, and document.all."

https://developer.mozilla.org/en-US/docs/Glossary/Truthy

The Ish in Three-ish

```
let obj: any = { x: 0 };
obj.foo();
obj();
obj.bar = 100;
obj = "hello";
const n: number = obj;

let obj: any {x:0, name:”Chris B. Behrens”};
```

One More Point on the -lsh

The primary types for TypeScript

A handful of pseudo-types

String, number, boolean

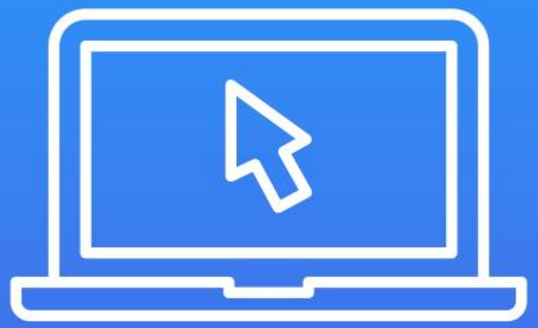
The Ish in Three-ish

```
const name: string = "Chris B. Behrens";
const knowsPython: boolean = true;
const salary: number = 100000000000;

calculateSalary(): number{
    return Math.pow(1000, 2);
}

calculateSalary(salary: number): number{
    return salary * 1.5;
}
```

Demo: TypeScript Typing vs. JavaScript Typing



Types in JavaScript

Types in TypeScript

How this all works when JavaScript talks

To TypeScript emitted JavaScript

The SensorId Property

```
Object.defineProperty(dataPacket.prototype, "sensorId", {
        get: function () {
            return this._sensorId;
        },
        set: function (value) {
            this._sensorId = value;
        },
        enumerable: false,
        configurable: true
    });
```

Another Word on the Any Type and Functions

I favor strict typing

Return types being optional is right

Exceptions make it even more complicated

Plus tuples...

Look at the lifetime of the variable carefully and work with a single type

Some More on Unit Testing and Types

If you want dynamic typing, there's a price in unit testing you're going to have to pay.

https://bit.ly/3tSFfhO

Jared Par on StackOverflow

There is one immutable fact about software quality. “If it can't compile, it can't ship.” In this rule, statically typed languages will win over dynamically typed languages.
Ok, yes, this rule is not immutable. Web Apps can ship without compiling (I've deployed many test web apps that didn't compile).

Jared Par

**But what is fundamentally true is “The sooner you catch an error, the cheaper it is to fix”
A statically typed language will prevent real errors from happening at one of the earliest possible moments in the software development cycle. A dynamic language will not. Unit Testing, if you are thorough to a superhuman level can take the place of a statically typed language.**

Jared Par

However, why bother? There are a lot of incredibly smart people out there writing an entire error checking system for you in the form of a Compiler. If you're concerned about getting errors sooner use a statically typed language.

The Next Step with This

You're the test developer

You're writing the same test over and over again

So, you would reinvent TypeScript

Consider the purpose of static type checking: avoiding a class of code defects (bugs). However, this has to be weighed in the context of the larger domain of all code defects. What matters most is not a comparison along a narrow sliver but a comparison across the depth and breadth of code quality, ease of writing correct code, etc.

If you can come up with a development style / process which enables your team to produce higher quality code more efficiently without static type checking, then it's worth it. This is true even in the case where you have holes in your testing that static type checking would catch.

I Already Bought the Car

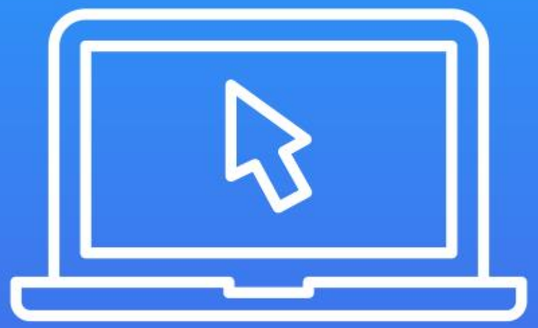
Why are you selling me on types?

Don't miss an opportunity

Avoid using "any" because you're under pressure

Get the ethos into your blood

Demo: An Avoided Unit Test



A JavaScript unit test that is necessary with loose typing

Refactor it to TypeScript

Take another look at the unit test

See that it's no longer necessary

Understand the class of unit testing that goes up in smoke with TypeScript

Wrap-up

There are limits to how far this goes

“Creating and Using Decorators in JavaScript”

A Few More Tricks with Types

Inference by Assignment

```
let age = 16;
console.log(typeof(age));

let age = “sixteen”;
console.log(typeof(age));
```

Custom Types

```
type bandMember = “bassist” | “vocalist” |
“drummer” | “lead guitarist” | “keyboardist”;

enum bandMember = {
Bass = “bassist”, Sing = “vocalist”, Drum =
“drummer”, Guitar = “lead guitarist”, Keys =
“keyboardist”
}
```

Summary

Type Inference and JavaScript Types

The TypeScript types

A demo of how they both work

The relationships of all this
with unit testing

A demo of migrating a Javascript type to TypeScript

How that can make code and unit tests go up in smoke