ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Overview & Speedrun Typescript

Alex Thalhammer

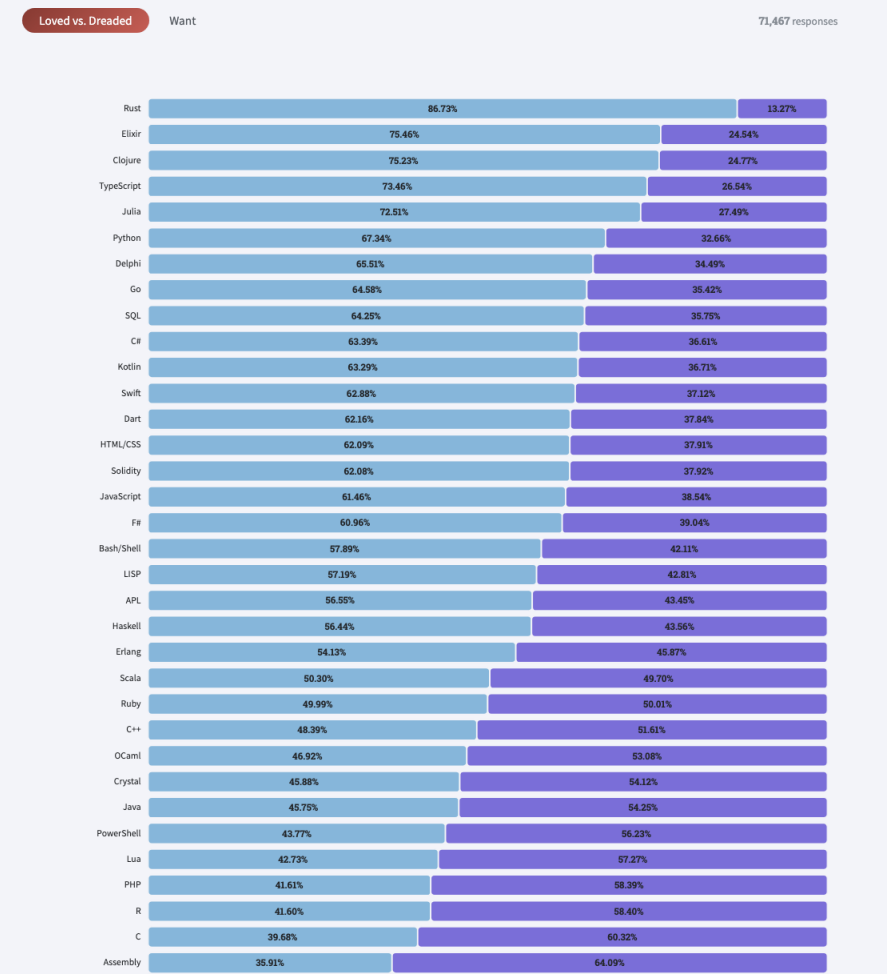
Overview

Page ▪ 2

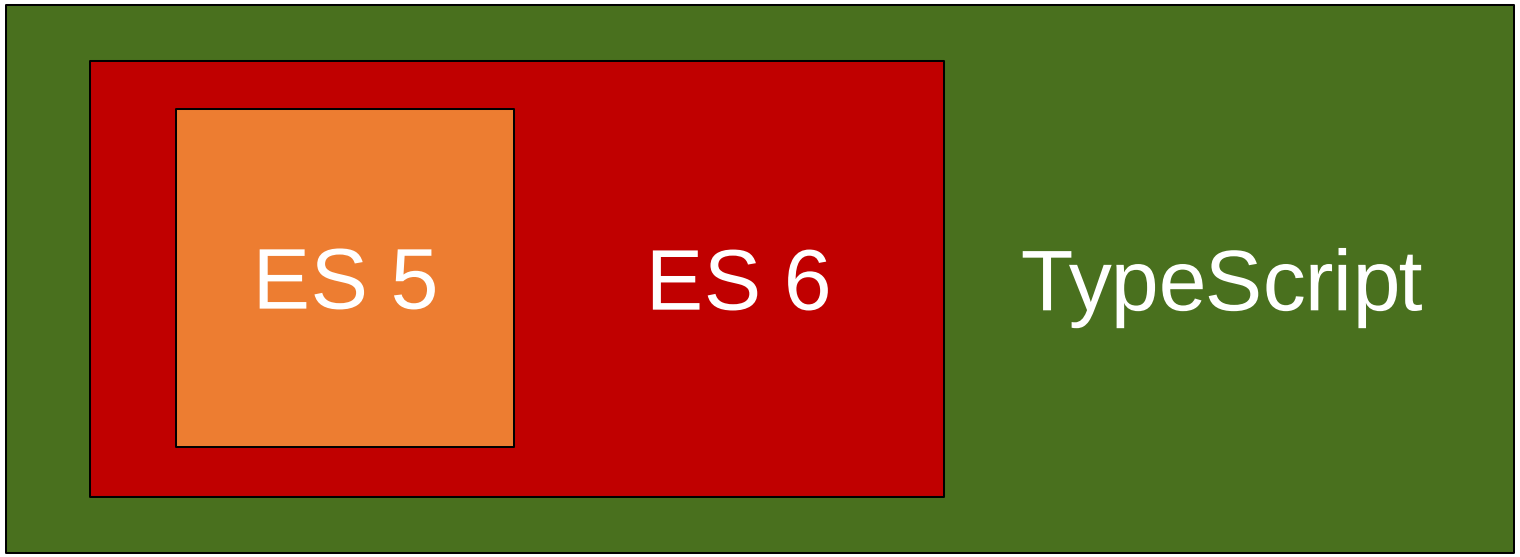
What is TypeScript?

- Superset of EcmaScript (JavaScript) 2022 or newer
- Compiles to EcmaScript 2022 or older (e.g. 2020, or 2015, or even 5 → IE11)
- Introduces static typing (like known from C++, C#, Java & other fancy languages)
- Advantages
  - Awesome Code Completion (IDE)
    - Easier refactoring <3
  - Static Code Analyzing (Compiler)
    - Faster detection of bugs ☺

Devs <3 TypeScript (StackOverflow Survey‘22)



TypeScript & ES6



compilation

How can we use TypeScript?

- Compiler in terminal / shell
- Integration in VS Code or IntelliJ/WebStorm
  - http://www.typescriptlang.org/
- Framework support
  - Angular **obligatory**
  - React optional
  - VueJS optional
  - & many others

Data types

number	string	boolean	any
unknown	undefined	null	function
object	[Class]	[Interface]	[Enum]
void	never		

Access Modifier

public

protected

private

readonly

public is standard

A small example

A small example

```
export class Flight {
    public id: number; // int + double
    protected from: string;
    private to: string;
    readonly date: string; // ISO date

    constructor(id: number) {
        this.id = id;
    }
}
```

A small example

```
export class Flight {
    id: number; // int + double
    protected from: string;
    private to: string;
    readonly date: string; // ISO date

    constructor(id: number) {
        this.id = id;
    }
}
```

A small example

```
export class Flight {
    id: number; // int + double
    protected from: string;
    private to: string;
    readonly date: string; // ISO date

    constructor(public id: number) {
        this.id = id;
    }
}
```

A small example

```
export class Flight {
    protected from: string;
    private to: string;
    readonly date: string; // ISO date

    constructor(public id: number) {}
}
```

Another example

```
// flight-manager.ts
import { Flight } from './flight';

export class FlightManager {
    constructor(private cache: Flight[]) {}

    search(from: string, to: string): Flight[] { [...] }

    get count(): number { return this.cache.length; }
    get flights(): Flight[] { return this.cache; }
    set flights(c: Flight[]): void { this.cache = c; }
}
```

Inheritance

```
export class ExtendedFlightManager extends FlightManager {

    constructor(cache: Flight[]) {
        super(cache);
    }

    // Overwrite methods
    search() {
        […]

        return super.search();
    }

}
```

Optionally

Project setup & ecosystem

Typical projekt structur

src	• Source code (TypeScript, HTML, SCSS)
dist	• Dist
node_modules	• Libs
package.json	• Pointer to Libs and Scripts
tsconfig.json	• Config of TypeScript-Compiler

NPM / Yarn / PNPM & package.json

package.json

```
{
    "dependencies": {
        "@angular/common": "2.0.0",
        […]
    },
    […]
}
```

package.json

~ tilde

```
{
    "dependencies": {
        "@angular/common": "~2.0.0",
        […]
    },
    […]
}
```

package.json

```
{
    "dependencies": {
        "@angular/common": "~2.0.0",
        […]
    },
    […]
}
```

package.json

^ caret

```
{
    "dependencies": {
        "@angular/common": "^2.0.0",
        […]
    },
    […]
}
```

package.json

```
{
    "dependencies": {
        "@angular/common": "^2.0.0",
        […]
    },
    […]
}
```

package.json

```
{
    "dependencies": {
        "@angular/common": "^2.0.0",
        […]
    },
    "devDependencies": {
       "webpack": "^1.12.9",
       "webpack-dev-server": "^1.14.0",
        […]
    },
    […]
}
```

package.json

```
{
    "dependencies": {
        "@angular/common": "^2.0.0",
        […]
    },
    "devDependencies": {
       "webpack": "^1.12.9",
       "webpack-dev-server": "^1.14.0",
        […]
    },
    "scripts": {
        "webpack": "webpack"
        "start": "webpack-dev-server",
    },
    […]
}
```

package.json

```
{
    "dependencies": {
        "@angular/common": "^2.0.0",
        […]
    },
    "devDependencies": {
       "webpack": "^1.12.9",
       "webpack-dev-server": "^1.14.0",
        […]
    },
    "scripts": {
        "webpack": "webpack --config webpack.dev.js"
        "start": "webpack-dev-server",
    },
    […]
}
```

package.json

```
{
    "dependencies": {
        "@angular/common": "^2.0.0",
        […]
    },
    "devDependencies": {
       "webpack": "^1.12.9",
       "webpack-dev-server": "^1.14.0",
        […]
    },
    "scripts": {
        "webpack": "webpack"
        "start": "webpack-dev-server",
    },
    […]
}
```

npm install or yarn or pnpm i

npm **run** webpack | yarn webpack

npm start or yarn start or pnpm start

TypeScript Speedrun ;-)

Types

Types

```
const name: string = "Max Muster";

let plz: number = 12345;

let autor: boolean = true;
```

Implicit types

```
const name = "Max Muster"; // string
let plz = 12345; // number
let autor = true; // boolean
```

Especially these 3 types!

Any

```
let website2: any = "http://www.softwarearchitekt.at";
website2 = 1;
```

Like JS ☺ → try to avoid as much as possible!

Unknown

```
let unknown: unknown;
unknown = 'unknown';
```

Better then any, still only if necessary!

Union-Types

```
let nameOrNumber: string | number;
nameOrNumber = "Max";
nameOrNumber = 17;
```

```
let strictVar: string | null | undefined;
let strictShrt?: string | null;
```

Union-Types as parameter

```
function showItem(speed: number | string) {
    if (typeof speed === 'number') {
        […]
    } else if (typeof speed === 'string') {
        […]
    }
}
```

Lambda statements

```
function filter(input: number[], callback: (item: number) => boolean):  number[] {
    const result: number[] = []; // new Array<number>();

    for (let i: number = 0; i < input.length; i++) {
        if (callback(objs[i])) {
            result.push(input[i]);
        }
    }

    return result;
}

let result = filter([1, 2, 3, 4, 5, 6], (item: number) => item % 2);
```

Falsy values

- undefined
- null
- false // boolean
- 0 // number
- '' // empty string
- NaN
  - NaN === NaN

Equal signs

- '1' == 1 → true
- false == 0 → true
  - inherently does type coercion
  - avoid this type-juggle

- '1' === 1 → false
- false === 0 → false
  - no type coercion
  - always use this 3 ☺

Error (Exception)

- `throw new Error("i'm an exception");`

Type Assertions

Type Assertions

```
let k: Contact = new Client(123, "Max Muster", "Essen");
```

Type Assertions

```
let k: Contact = new Client(123, "Max Muster", "Essen");

[…]
let art = k.clientAttr;  // won‘t work
```

Type Casting

```
let k: Contact = new Client(123, "Max Muster", "Essen");

[…]
let client = k as Client;
let art = client.clientAttr;  // OK
```

Type Casting (Alternative)

```
let k: Contact = new Client(123, "Max Muster", "Essen");

[…]
let client = <Client>k;
let art = client.clientAttr;  // OK
```

Interfaces

```
interface Contact {
    id: number;
    name: string;
    location?: string;
    plz: number;
    date: Date;
    getInfo(): string;
}
```

```
class MyContact implements Contact {
    […]
}
```

Interfaces

```
interface Contact {
    id: number;
    name: string;
    location?: string;
    plz: number | null;
    date?: Date | null;
}
```

used for model from backend!

Interfaces

```
interface Contact {
    id: number;
    name: string;
    location?: string;
    plz: number | null;
    date?: Date | null;
}
```

the *TypeScript* way

```
type Contact = {
    id: number;
    name: string;
    location?: string;
    plz: number | null;
    date?: Date | null;
};
```

Generics

```
export class ReadOnly<T> {
    private data: T;
    constructor(data: T) {
        this.data = data;
    }
    public getData(): T {
        return this.data;
    }
}

let readOnlyNumber = new ReadOnly<number>(42);
console.debug(readOnlyNumber.getData());
```

Functions

Optional Parameters

```
function sayHello(name = "noname"): void {
    console.debug("Hallo " + name);
}

sayHello("Max");
sayHello();
```

Optional Parameters

```
function sayHello(name?: string): void {
    if (name) {
        console.debug("Hallo " + name);
    } else {
        console.debug("Hallo!");
    }
}
```

Enums

Enums

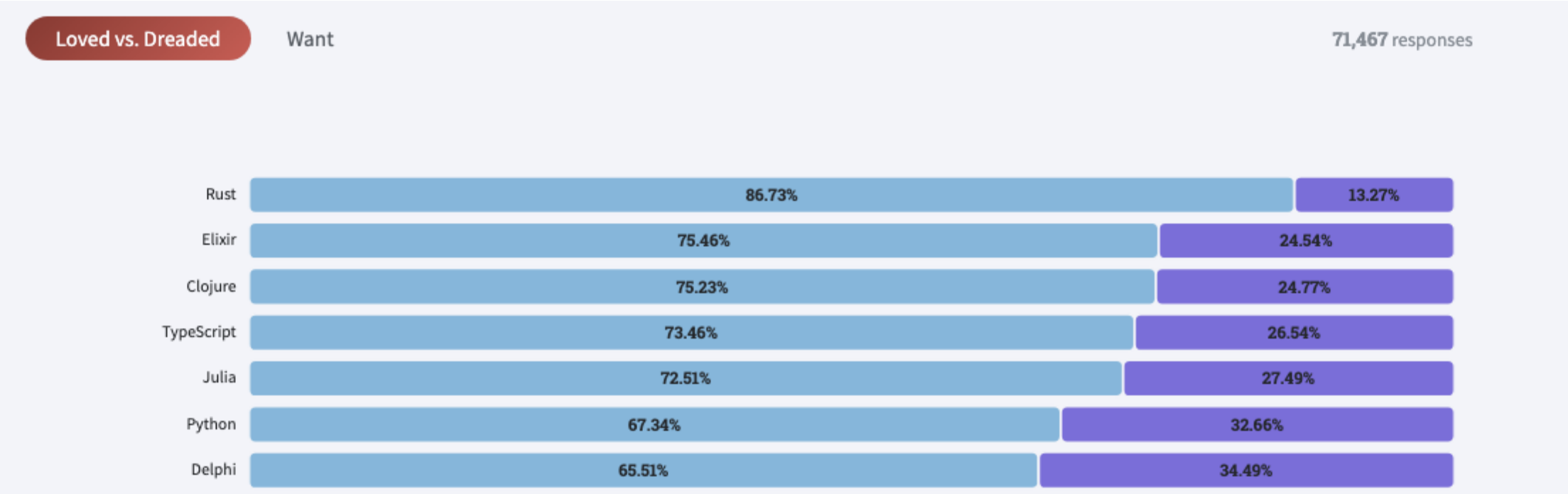
- enum Direction { UP, DOWN, LEFT, RIGHT };
  let d = Direction.UP;
- enum Dir { UP = 'up', DOWN = 'down', LEFT = 'left', RIGHT = 'right'};

- type Direction = 'UP' | 'DOWN' | 'LEFT' | 'RIGHT';
  let d: Direction = 'UP';

the *TypeScript* way

Now you know TypeScript

- Use it and you're gonna love it
  - Well, if you have used JS before ☺



Recommendations on TypeScript

Use prettier!

Use ESLint

- Avoid any!
- Use strict typing!
- Except inferred types ☺
- Also for functions params and return types!