

# TypeScript

JavaScript on steroids for applications at scale

Imola, 2019-05-16





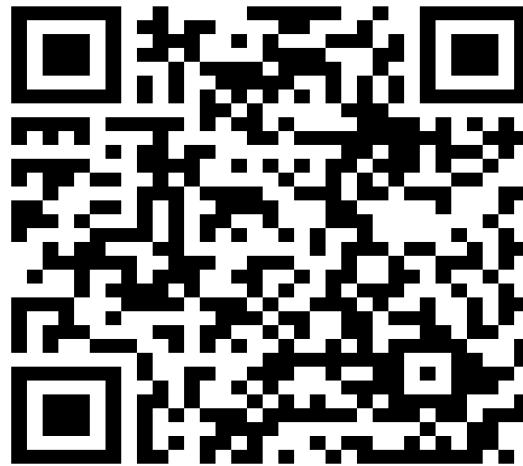
# Massimo Artizzu

Web dev & architect  
at [Antreem](#)

 /   
**@MaxArt2501**



# You can find these slides at



[maxart2501.github.io/typescript-talk/devromagna/](https://maxart2501.github.io/typescript-talk/devromagna/)

# **JavaScript** is weakly typed



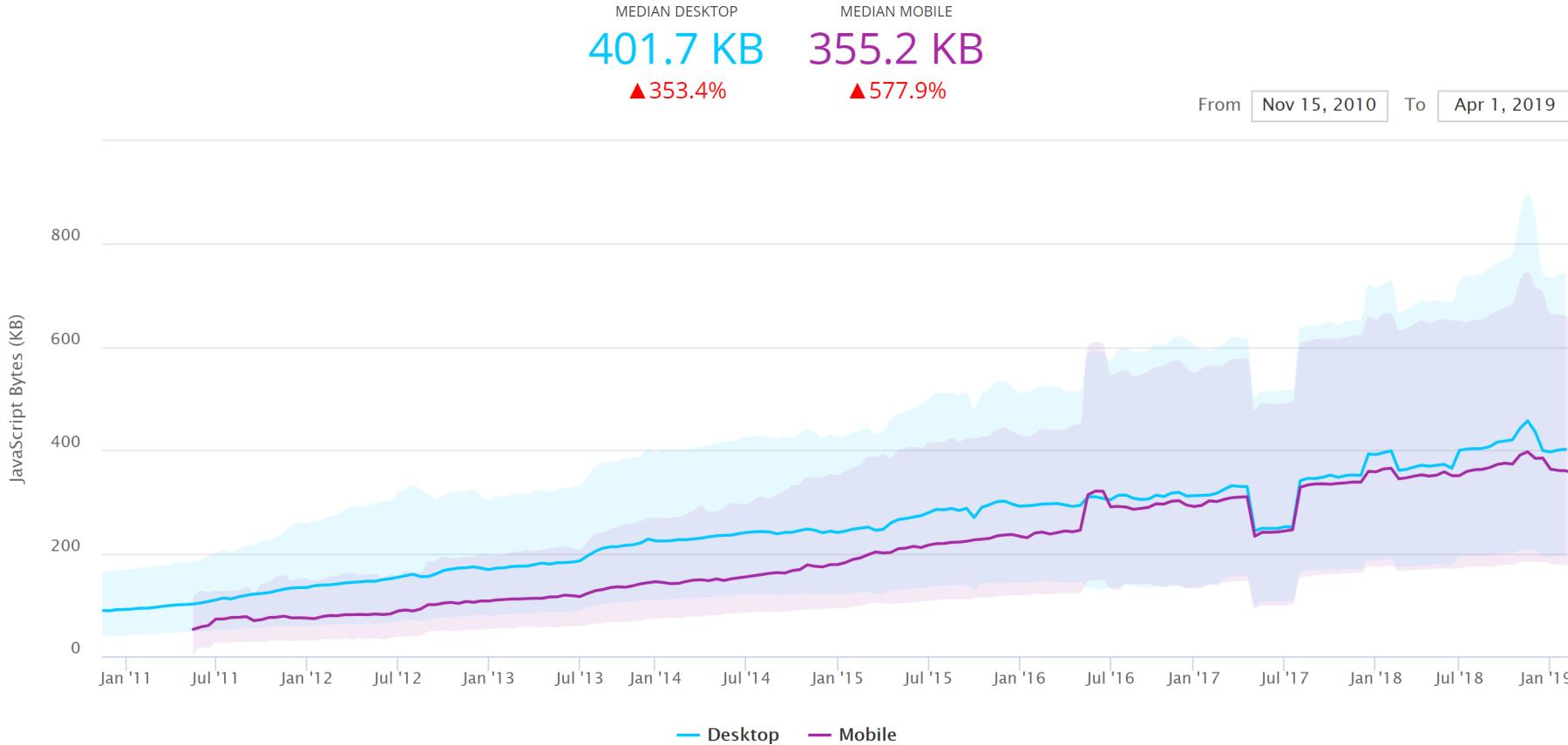
# ... because it's not *Java*

```
597 @  
598     private static VariableDeclarationImpl getVariableDeclaration(PsiVariable element, Document document) {  
599         if (element.getName() != null && element.getTypeElement() != null  
600             && (element.getInitializer() != null || element.getParent() instanceof PsiForEach))  
601             && element.getTextRange().getStartOffset() < element.getTypeElement().getTextRange().getEndOffset())  
602             boolean isFinal = calculateIfFinal(element);  
603             return new VariableDeclarationImpl(TextRange.create(  
604                 element.getTextRange().getStartOffset(),  
605                 element.getTypeElement().getTextRange().getEndOffset()),  
606                 isFinal: element.getModifierList() != null && isFinal);  
607         }  
608     }  
609     return null;  
610 }
```



**So... TypeScript.**





Source: <httparchive>



# **The problems we're facing with modern web development**

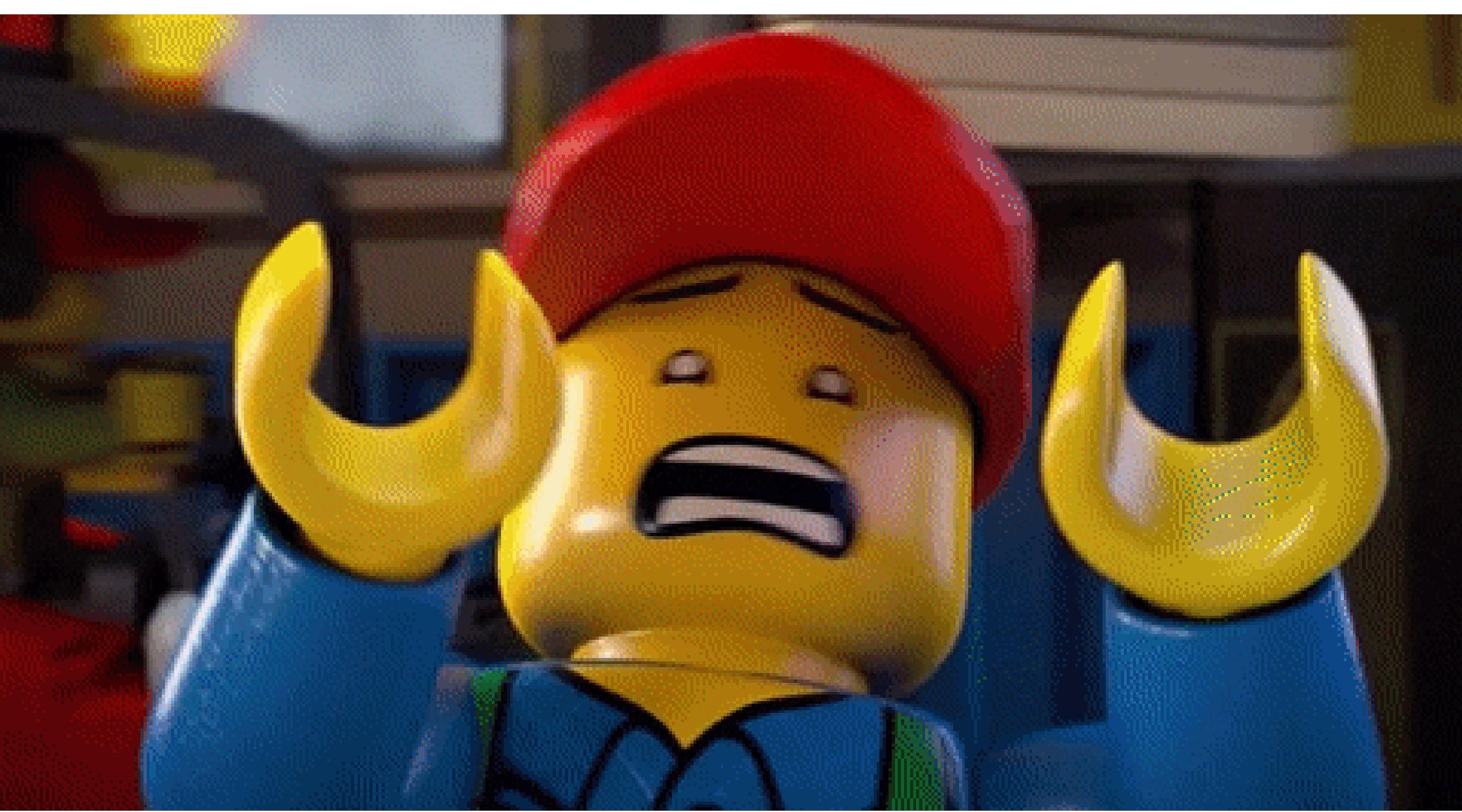
```
let total = costs.reduce(0, (num, sum) => num + sum);

const answer = list.lastFind(item => item.id === 42);

$.ajax({
  url: '...', method: 'POST',
  body: {...}, contentType: 'json'
});
```

**What happens  
with code just the team knows about?**

```
function addToCart(🛒, 🙄) {  
    ...  
}  
  
const car = new Car(🚗);  
car.engine = '⚡';  
car.setEngine('⚡');  
  
config.apiRot = '🌐';
```



```
enum Bool { TRUE, FALSE, FILE_NOT_FOUND };
```

```
import 'dart:io';
```

```
type
```

```
dex: number;
```

```
ST'.
```

```
internal user {
```

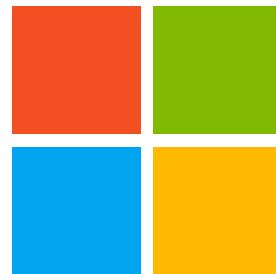
```
pri:
```

ty.pescriptlang.org

(with some caveats...)



# What's TypeScript?



# Microsoft



# JavaScript plus...

 types

 OOP concepts

 stage 3+ features



# Getting started

Try everything on the spot!

## 1. From npm

```
$ npm i -g typescript  
$ tsc my-module.ts --out my-module.js
```

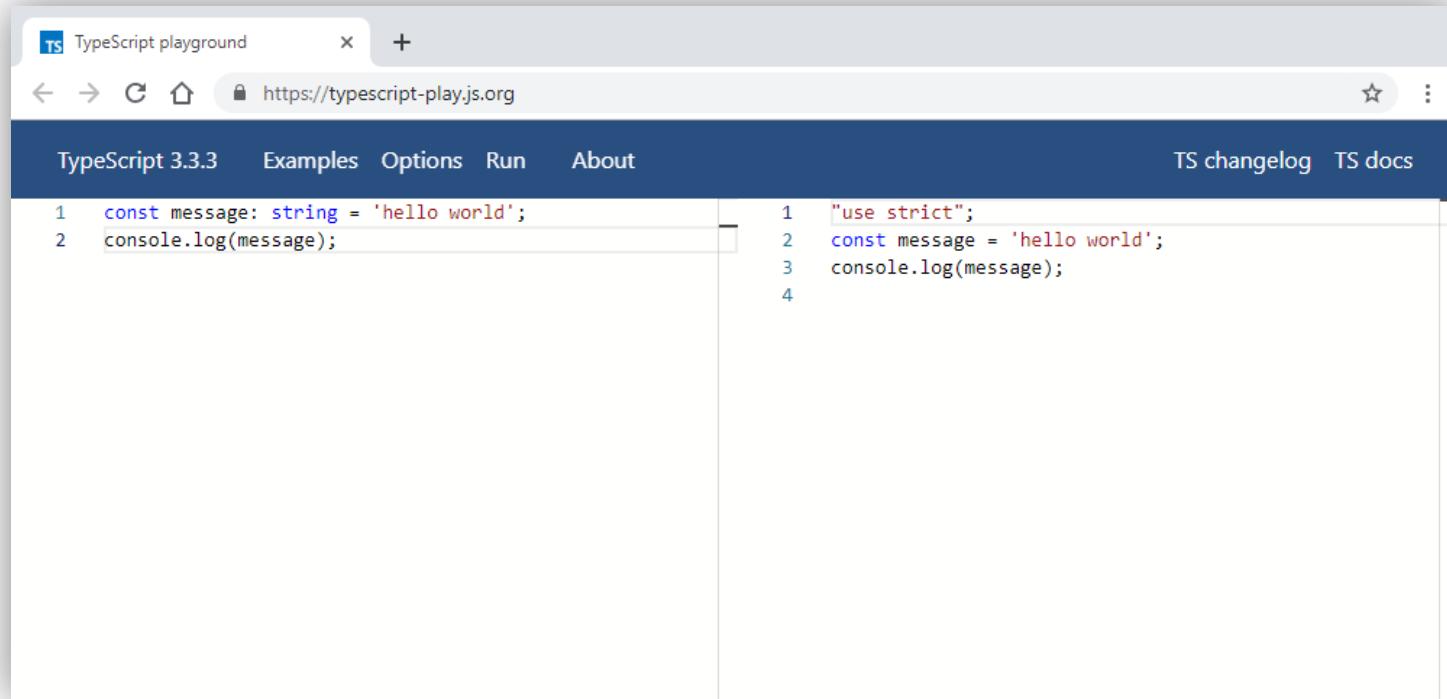
## 2. ts-node

```
$ npm i -g ts-node
```

Use it just like node

```
$ ts-node my-module.ts
```

# 3. On the web

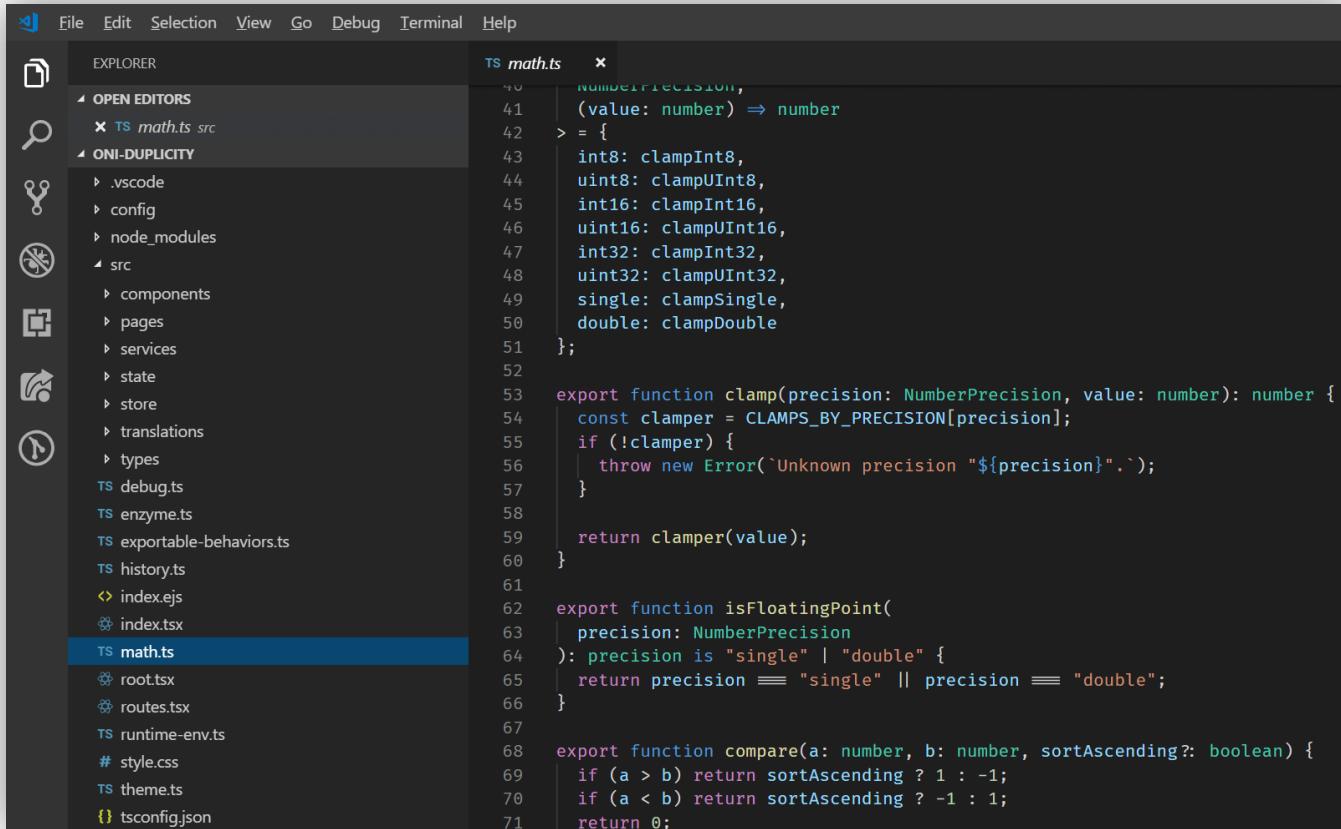


A screenshot of a web browser window titled "TypeScript playground". The URL in the address bar is <https://typescript-play.js.org>. The page has a dark blue header with the text "TypeScript 3.3.3 Examples Options Run About" on the left and "TS changelog TS docs" on the right. Below the header, there are two code snippets. The left snippet shows a "hello world" program in TypeScript:1 const message: string = 'hello world';
2 console.log(message);The right snippet shows the same code after it has been transpiled to JavaScript:1 "use strict";
2 const message = 'hello world';
3 console.log(message);
4



[typescript-play.js.org](https://typescript-play.js.org)

# 4. In your IDE



The screenshot shows the Visual Studio Code interface. The left sidebar is the Explorer view, displaying a tree structure of files and folders. The 'math.ts' file is selected in the tree, indicated by a blue highlight. The main editor area shows the code for the 'math.ts' file. The code defines several utility functions for number precision:

```
ts math.ts
  40 |     NUMBERPRECISION,
  41 |     (value: number) => number
  42 |   > = {
  43 |     int8: clampInt8,
  44 |     uint8: clampUInt8,
  45 |     int16: clampInt16,
  46 |     uint16: clampUInt16,
  47 |     int32: clampInt32,
  48 |     uint32: clampUInt32,
  49 |     single: clampSingle,
  50 |     double: clampDouble
  51 |   };
  52 |
  53 | export function clamp(precision: NumberPrecision, value: number): number {
  54 |   const clamper = CLAMPS_BY_PRECISION[precision];
  55 |   if (!clamper) {
  56 |     throw new Error(`Unknown precision "${precision}"`);
  57 |   }
  58 |
  59 |   return clamper(value);
  60 | }
  61 |
  62 | export function isFloatingPoint(
  63 |   precision: NumberPrecision
  64 | ): precision is "single" | "double" {
  65 |   return precision === "single" || precision === "double";
  66 | }
  67 |
  68 | export function compare(a: number, b: number, sortAscending?: boolean) {
  69 |   if (a > b) return sortAscending ? 1 : -1;
  70 |   if (a < b) return sortAscending ? -1 : 1;
  71 |   return 0;
}
```

# Using types

```
let answer: number = 42;  
let name: string = 'Ford Prefect';  
let isAlien: boolean = true;  
let entity: symbol = Symbol('question');  
let whatever: any;
```

# Arrays

```
let fib: number[]          = [ 0, 1, 1, 2, 3, 5 ];  
let fib: Array<number> = [ 0, 1, 1, 2, 3, 5 ];  
  
let point: [ number, number ] = [ 2, -1 ];
```

# Objects

```
let user: {  
    name: string,  
    email?: number  
} = { name: 'Emma' };  
  
let boundaries: {  
    [key: string]: number  
} = { min: 0, max: 100 };
```

# Functions

```
function count(src: string, ref: string): number {  
    return source.split(ref).length - 1;  
}
```

```
let minFn: (...items: string[]) => string;
```

# Generics

```
interface PagedList<T> {  
    items: T[];  
    total: number;  
}  
  
function getResource<T>(url: string): PagedList<T>
```

# Notable cases

```
const roles: Array<string> =  
  [ 'USER', 'ADMIN', 'GUEST' ]  
  
let request: Promise<Request> = fetch('...')  
let models: Promise<Model[]> = request.json()  
  
let user$: Observable<User>
```

# Type inference

```
let answer = 42                      // => number
let states = ['on', 'off']            // => string[]
let user = {name: 'Emma'}           // => {name: string}
let none = null                     // => any
function count(...) { ... }
    // (src: string, ref: string) => number
```

# A case for functions

```
function getCars() {  
    return getList<Car>('...');  
}  
// () => PagedList<Car>  
getCars().  
    ▾ items      (property) Car[]  
    ▾ total
```

```
function getCars(): Car[] {  
    return getList<Car>('...');  
}
```

Type 'PagedList<Car>' is  
not assignable to type  
'Car[]'. (ts2322)

# Type aliases

```
type Point2D = [ number, number ]
type User = { id: number, name: string }
type Method = 'GET' | 'POST' | 'DELETE'
type Files = HTMLInputElement['files'] // File[]
type CountFn = typeof count
```

# What defines a type? 🤔

```
type Point2D = [ number, number ];
```

```
interface User { ... }
```

```
class Button { ... }
```

```
enum LogLevel { WARN, ... }
```

# Why **interfaces**? 🧑 (and not **types**)

"Type aliases cannot be extended or implemented from (nor can they extend/implement other types)"

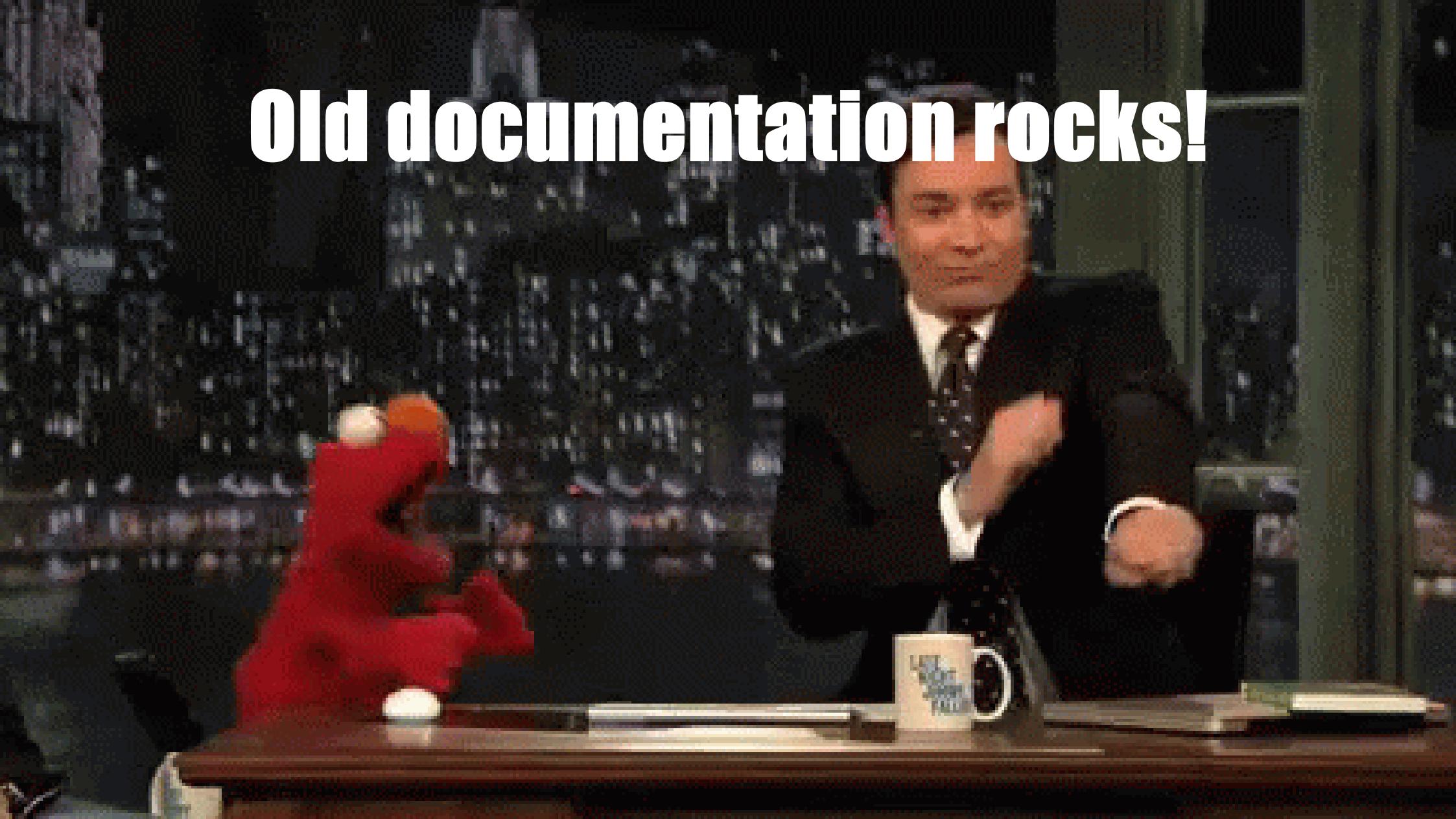
```
interface Loggable {  
    username: string;  
    login(): boolean;  
}  
  
class User  
    implements Loggable  
{ ... }
```

```
type Loggable = {  
    username: string;  
    login(): boolean;  
}  
  
class User  
    implements Loggable  
{ ... }
```

```
interface Admin extends Loggable {  
    addUser(user: User): boolean;  
}
```

```
type SuperAdmin = Admin & {  
    deleteUser(user: User): boolean;  
}
```

# Old documentation rocks!



## **type** aliases

## **interface**



type shortenings and  
manipulations



object *shape* definitions



# Models

```
// models/car.ts  
  
interface Car {  
    model: string;  
    brand: Brand;  
    engine: Engine;  
}
```

```
// models/engine.ts  
  
interface Engine {  
    model: string;  
    fuel: FuelType;  
    size?: number;  
}
```

```
// models/brand.ts  
  
interface Brand {  
    name: string;  
}
```

```
// models/fuel-type.ts  
  
enum FuelType {  
    GASOLINE, DIESEL  
}
```



```
class Brand {  
  name: string;  
  constructor(name: string) {  
    this.name = name;  
  }  
}
```



```
interface Brand {  
  name: string;  
}  
  
const brand: Brand = {  
  name: 'Toyota'  
};
```



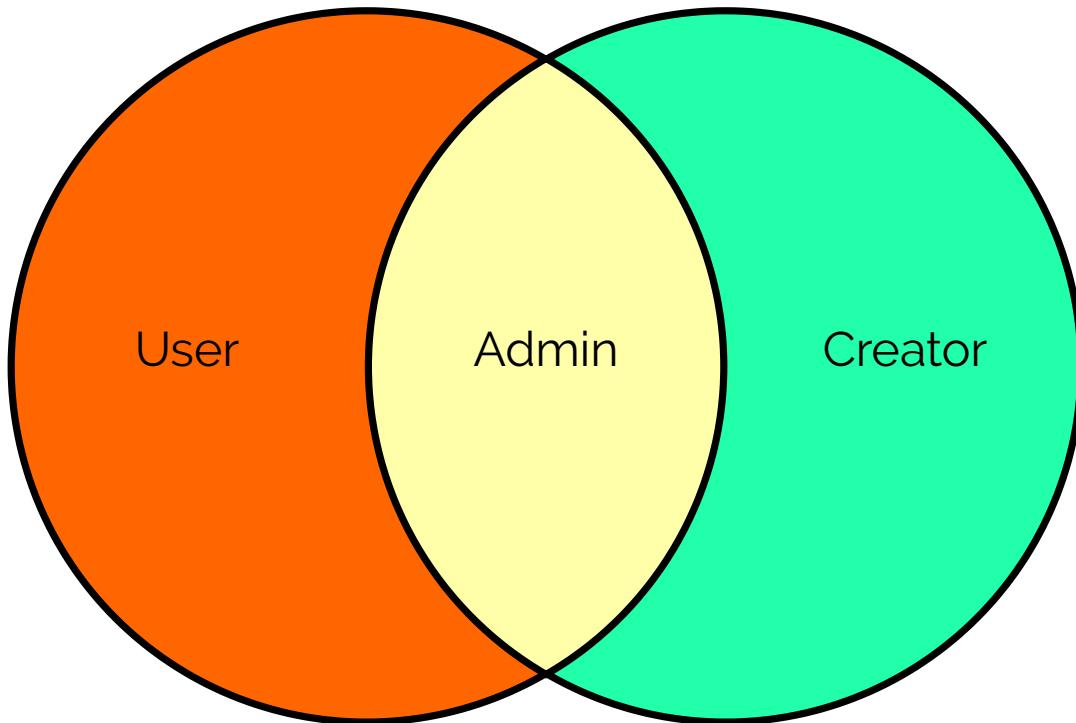
# Operations on types

# Type intersection

```
type User = { username: string; }

type Creator = {
    addUser: (username: string) => boolean;
}

type Admin = User & Creator;
```



# Type union

```
function utter(answer: string | number) {  
  console.log(`The answer is ${answer}`);  
}  
  
class Family {  
  pet: Cat | Dog;  
}  
  
type Method = 'GET' | 'POST' | 'DELETE';
```



# Type guards

```
function getValue(  
  element: HTMLInputElement | HTMLSelectElement  
) {  
  if (element.matches('.my-select')) {  
    const index = element.selectedIndex;  
    return element.options[index].text;  
  }  
  return element.va  
}
```

Property 'options' does not exist on type 'HTMLInputElement | HTMLSelectElement'.

Property 'options' does not exist on type 'HTMLInputElement'.

```
const index = (<HTMLSelectElement>element)
    .selectedIndex;
return (element as HTMLSelectElement)
    .options[index].text;
```

```
if (element instanceof HTMLSelectElement) {
  const index = element.selectedIndex;
  return element.options[index].text;
}
```

# Kinds of type guards

```
if (typeof value === 'string') { ... }
```

```
if (ref instanceof HTMLElement) { ... }
```

```
if ('value' in item) { ... }
```



# Tagged unions

```
interface Ticket {  
    kind: 'TICKET';  
    quantity: number;  
}  
  
interface Seasonal {  
    kind: 'SEASONAL';  
    price: number;  
}
```

```
type Item =  
    | Ticket  
    | Seasonal  
  
type Kinds  
    = Item['kind']  
    // 'TICKET' | 'SEASONAL'
```

```
function getPrice(item: Item) {  
  if (item.kind === 'TICKET') {  
    return item.quantity * TICKET_PRICE;  
  }  
  return item.price;  
}
```

# Custom type guards

```
function isTicket(item: Item): item is Ticket {  
    return item.kind === 'TICKET';  
}  
  
function getPrice(item: Item) {  
    if (isTicket(item)) {  
        return item.quantity * TICKET_PRICE;  
    }  
    return item.price;  
}
```



# Conditional types

T extends U ? X : Y

```
type Exclude<T, U> = T extends U ? never : T
```

```
type NonNullable<T> = Exclude<T, null | undefined>
```

```
interface UserWithId {  
    id: string;  
    name: string;  
}  
  
type UserKeys = keyof UserWithId; // 'id' | 'name'  
  
type User = {  
    [K in Exclude<UserKeys, 'id'>]: UserWithId[K];  
}; // { name: string }
```

```
type MethodNames<T> = {  
  [K in keyof T]: T[K] extends Function ? K : never;  
}[keyof T];
```

```
interface Dog {  
  name: string;  
  bark: () => string;  
  walk: (mins: number) => void;  
}  
type DogActions = MethodNames<Dog>; // 'bark' | 'walk'
```

```
interface IndexAction {  
    kind: 'INDEX';  
    index: number;  
}  
  
interface TitleAction {  
    kind: 'TITLE';  
    title: string;  
}  
  
type Action = IndexAction | TitleAction;
```

```
type GetFromKind<A, T>
= A extends { kind: T } ? A : never;
```

```
type ActionFromKind<T extends Action['kind']>
= GetFromKind<Action, T>;
```

```
ActionFromKind<'INDEX'> // IndexAction
```



**Where do all those definitions come  
from?**

```
// tsconfig.json
"compilerOptions": {
    ...
    "lib": [ "es2018", "dom" ]
}
```

 node\_modules

 axios

 changelog.md

 index.d.ts

 package.json

...

```
// packages.json  
...  
"typings": "types.d.ts"
```

**What about other packages?**

definitelytyped.org

```
npm i -D @types/jquery
```

```
// tsconfig.json
"compilerOptions": {
    ...
    "typeRoots": [ "node_modules/@types" ]
}
```

# FUNCTIONAL PROGRAMMING POWERS!





[github.com/gcanti/fp-ts](https://github.com/gcanti/fp-ts)

```
const optionalUser: Option<User>
  = fromNullable(user);
const userName = optionalUser
  .map(user => `${user.name} ${user.surname}`)
  .getOrElse('-');
```

Linting? 🤔

# TSLint

no-any no-empty-interface no-inferable-types no-internal-module no-non-null-assertion no-unnecessary-type-assertion no-var-requires typedef unified-signatures await-promise no-misused-new no-unsafe-any no-unused-variable prefer-readonly interface-over-type-literal no-angle-bracket-type-assertion ...

**... or maybe ESLint + TypeScript?**



# TypeScript compiler options

## --strictNullChecks

```
let index: number = null;
```

```
let index: number
```

```
Type 'null' is not assignable  
to type 'number'. (ts2322)
```

```
function getMailLink(user: User) {  
    return `mailto:${user.email}`;  
}
```

```
getMailLink(null); // BOOM
```

## --noImplicitAny

```
function sum(list) { ... }
```

```
class MainComponent {  
    index:  
}
```

(property) MainComponent.index: any  
Member 'foo' implicitly has an 'any'  
type. (ts7008)

--strict

--noImplicitAny

--noImplicitThis

--alwaysStrict

--strictBindCallApply

--strictNullChecks

--strictFunctionTypes

--strictPropertyInitialization

--target

ES3

ES5

ES2015

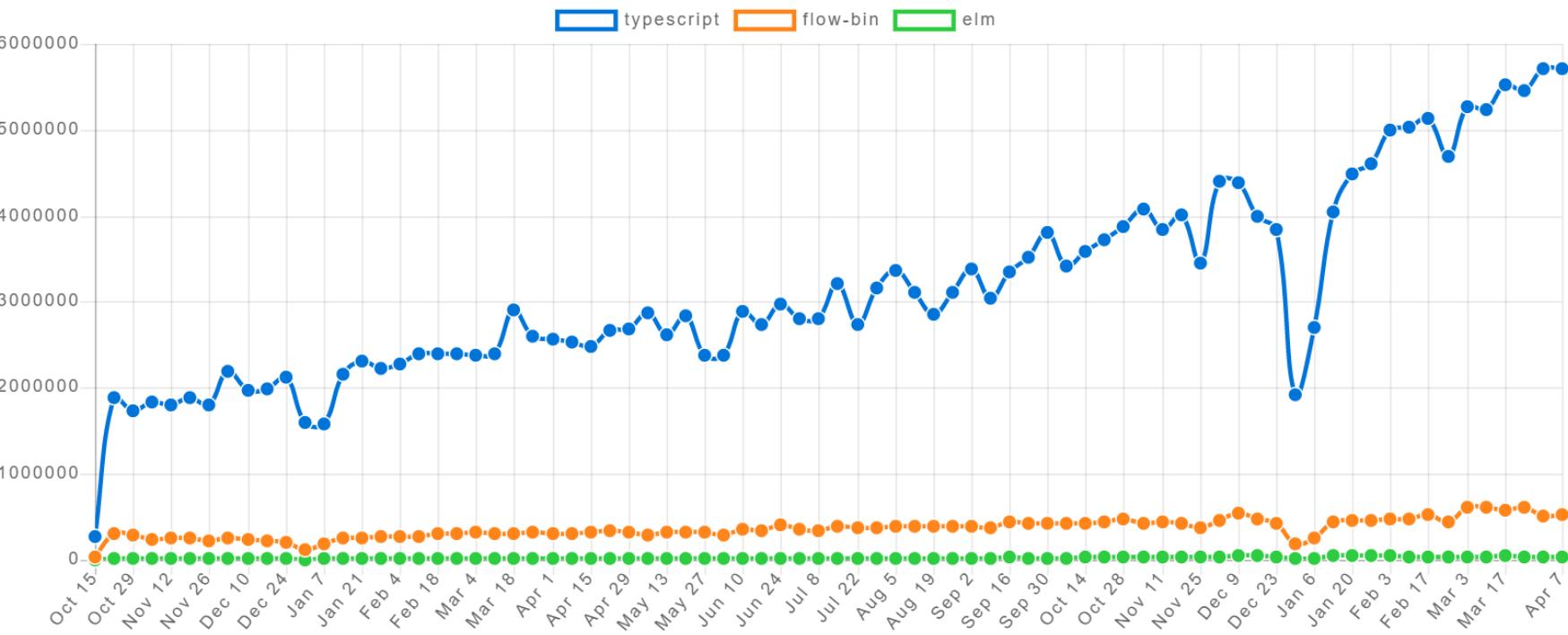
ES2016

ES2017

ES2018

ESNext

[typescriptlang.org/docs/handbook/compiler-options.html](https://typescriptlang.org/docs/handbook/compiler-options.html)



Source: [npmtrends.com](http://npmtrends.com)

# Bad reasons

to use TypeScript

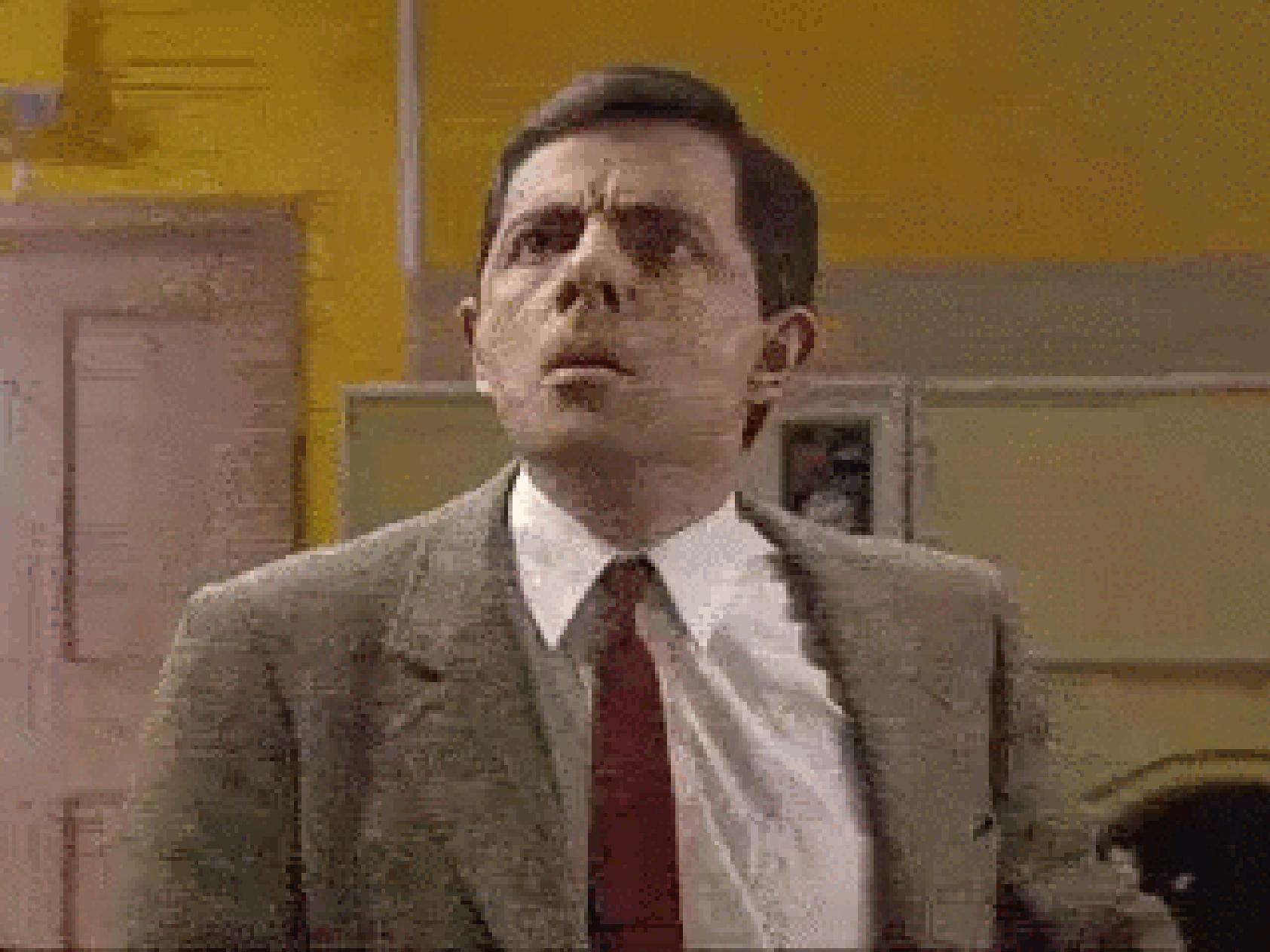
**"It looks more like Java/C#, so I don't  
have to learn JavaScript"**

A dramatic, close-up shot of a man with long, dark hair and a beard. He is wearing a dark, textured cloak. His right hand is raised to his forehead, with his fingers resting against his temple in a gesture of distress, exhaustion, or despair. The lighting is low and warm, casting deep shadows and highlighting the contours of his face and hair. The background is blurred, showing what appears to be a stone wall or doorway.

**One does not simply ignore JavaScript**

**"No more unit tests!"**

**"Only" 15% of bugs are type related and could be prevented by TypeScript or other static typing systems (2017).**



**"My code won't have runtime type  
problems anymore"**

```
function getUsers(): Promise<User[]> {
  return fetch('...')
    .then(response => response.json());
}
```

```
function  
{  
    console  
}  
)
```



```
+
```

# Links

- 🔗 Interface vs Type alias in TypeScript 2.7 @martin\_hotell  
[medium.com/@martin\\_hotell/interface-vs-type-alias-in-typescript-2-7-2a8f1777af4c](https://medium.com/@martin_hotell/interface-vs-type-alias-in-typescript-2-7-2a8f1777af4c)
- 🔗 TypeScript 2.8: Conditional Types @mariusschulz  
[mariusschulz.com/blog/typescript-2-8-conditional-types](https://mariusschulz.com/blog/typescript-2-8-conditional-types)
- 🔗 Why use TypeScript, good and bad reasons @whereischarly  
[itnext.io/why-use-typescript-good-and-bad-reasons-ccd807b292fb](https://itnext.io/why-use-typescript-good-and-bad-reasons-ccd807b292fb)
- 🔗 Italia JS Slack #typescript channel  
[italiajs.slack.com/messages/CAHLHRZAQ/](https://italiajs.slack.com/messages/CAHLHRZAQ/)

**That's all, folks!**



[maxart2501.github.io/typescript-talk/devromagna/](https://maxart2501.github.io/typescript-talk/devromagna/)



```
const question: Question[] = await getQuestions()
questions.forEach((question: Question) => {
  question.answer()
})
```