TypeScript

JavaScript on steroids for applications at scale

Imola, 2019-05-16

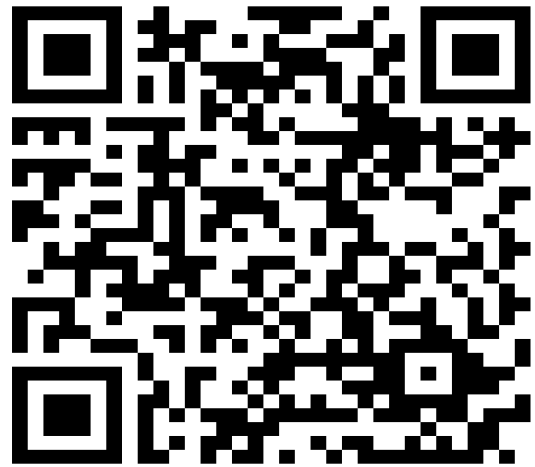
Massimo Artizzu

Web dev & architect
at Antreem

/

@MaxArt2501

You can find these slides at



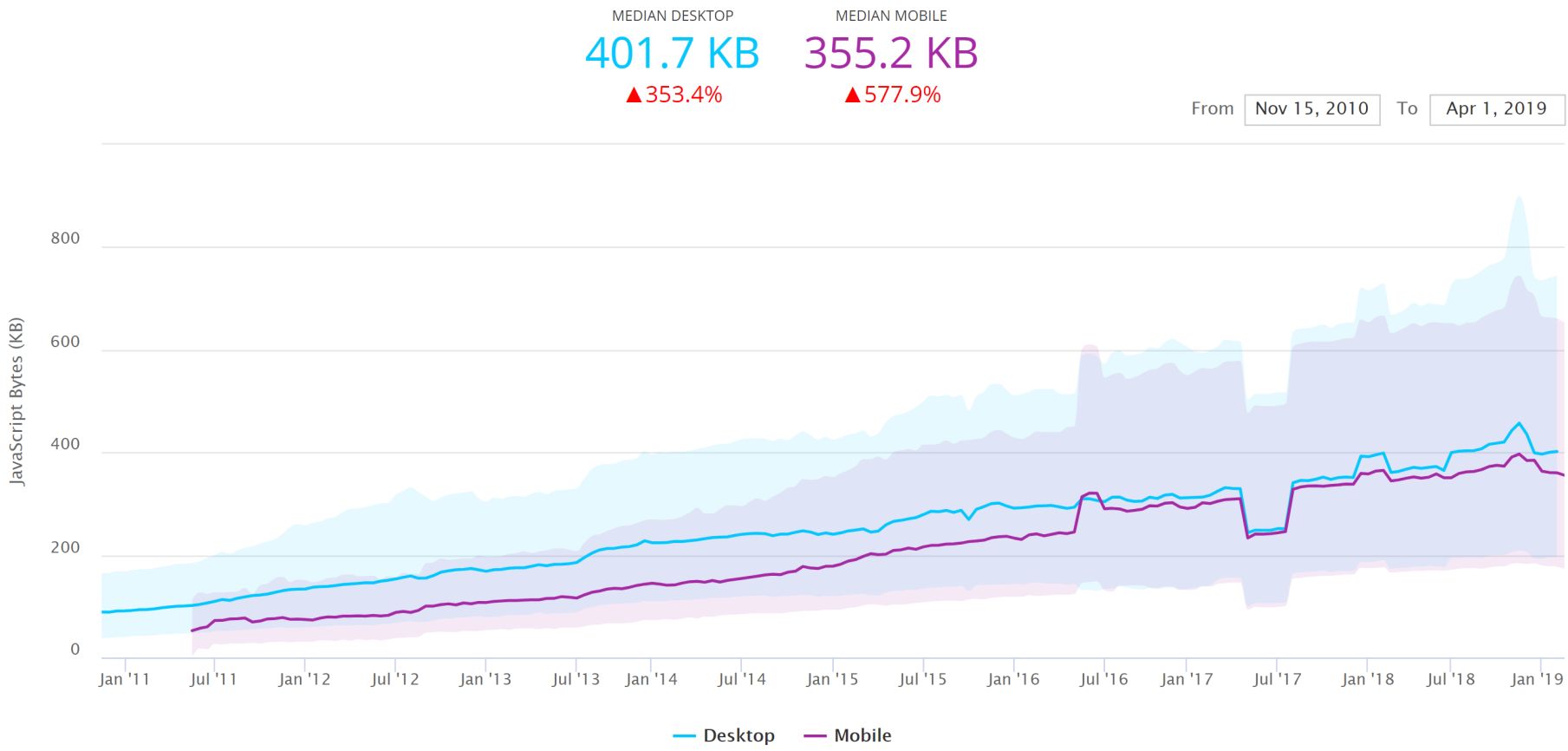
maxart2501.github.io/typescript-talk/devromagna/

JavaScript is weakly typed

… because it's not *Java*

```
597
598 @   private static VariableDeclarationImpl getVariableDeclaration(PsiVariable element, Document g
599         if (element.getName() ≠ null && element.getTypeElement() ≠ null
600                 && (element.getInitializer() ≠ null || element.getParent() instanceof PsiForeach
601                 && element.getTextRange().getStartOffset() < element.getTypeElement().getTextRang
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         return null;
609     }
```

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 };

typescriptlang.org

(with some caveats...)

What's TypeScript?


Microsoft


TS
C
JS

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
```

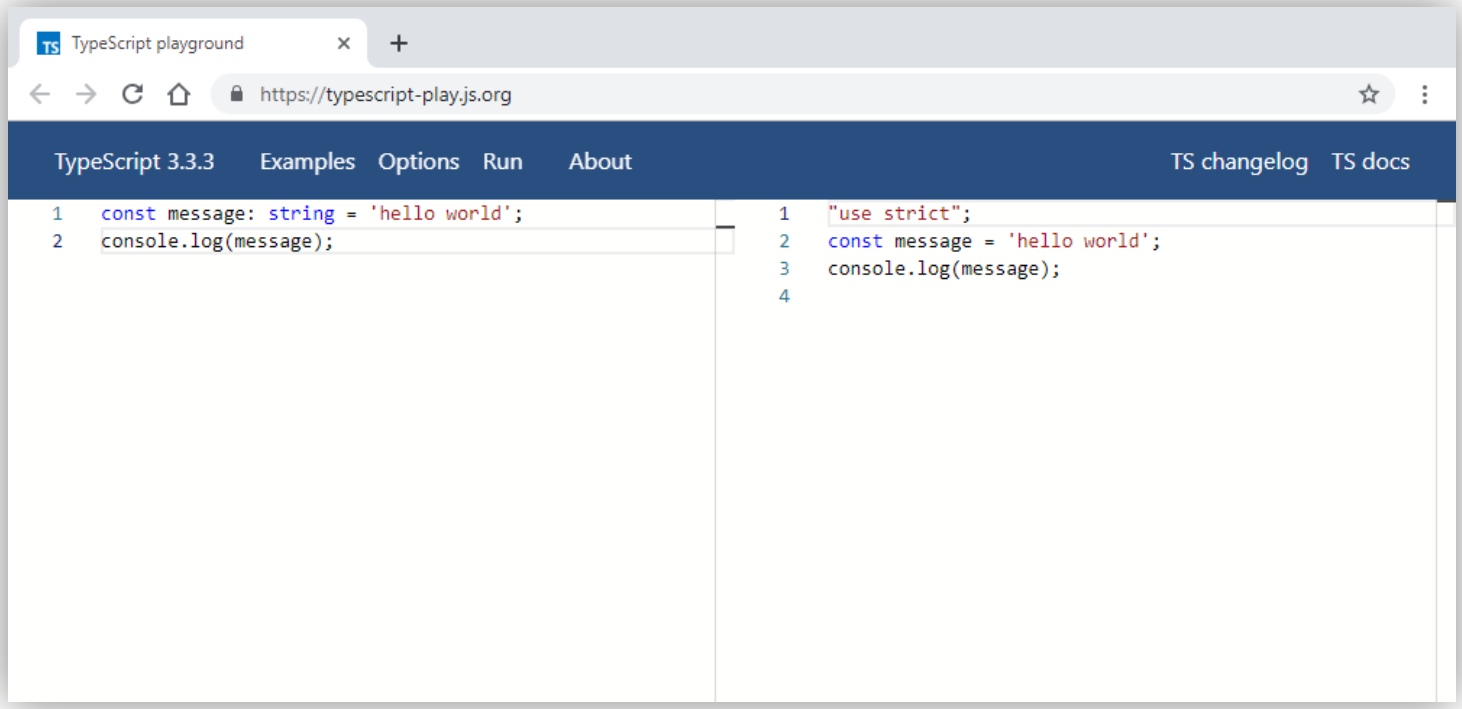
2. `ts-node`

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

Use it just like `node`

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

3. On the web



typescript-play.js.org

4. In your IDE

File Edit Selection View Go Debug Terminal Help

EXPLORER

OPEN EDITORS
- math.ts src

ONI-DUPLICITY
- .vscode
- config
- node_modules
- src
  - components
  - pages
  - services
  - state
  - store
  - translations
  - types
  - debug.ts
  - enzyme.ts
  - exportable-behaviors.ts
  - history.ts
  - index.ejs
  - index.tsx
  - math.ts
  - root.tsx
  - routes.tsx
  - runtime-env.ts
  - style.css
  - theme.ts
  - tsconfig.json

math.ts

```
  NumberPrecision,
  (value: number) => number
> = {
  int8: clampInt8,
  uint8: clampUInt8,
  int16: clampInt16,
  uint16: clampUInt16,
  int32: clampInt32,
  uint32: clampUInt32,
  single: clampSingle,
  double: clampDouble
};

export function clamp(precision: NumberPrecision, value: number): number {
  const clamper = CLAMPS_BY_PRECISION[precision];
  if (!clamper) {
    throw new Error(`Unknown precision "${precision}".`);
  }

  return clamper(value);
}

export function isFloatingPoint(
  precision: NumberPrecision
): precision is "single" | "double" {
  return precision === "single" || precision === "double";
}

export function compare(a: number, b: number, sortAscending?: boolean) {
  if (a > b) return sortAscending ? 1 : -1;
  if (a < b) return sortAscending ? -1 : 1;
  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, ... }
```

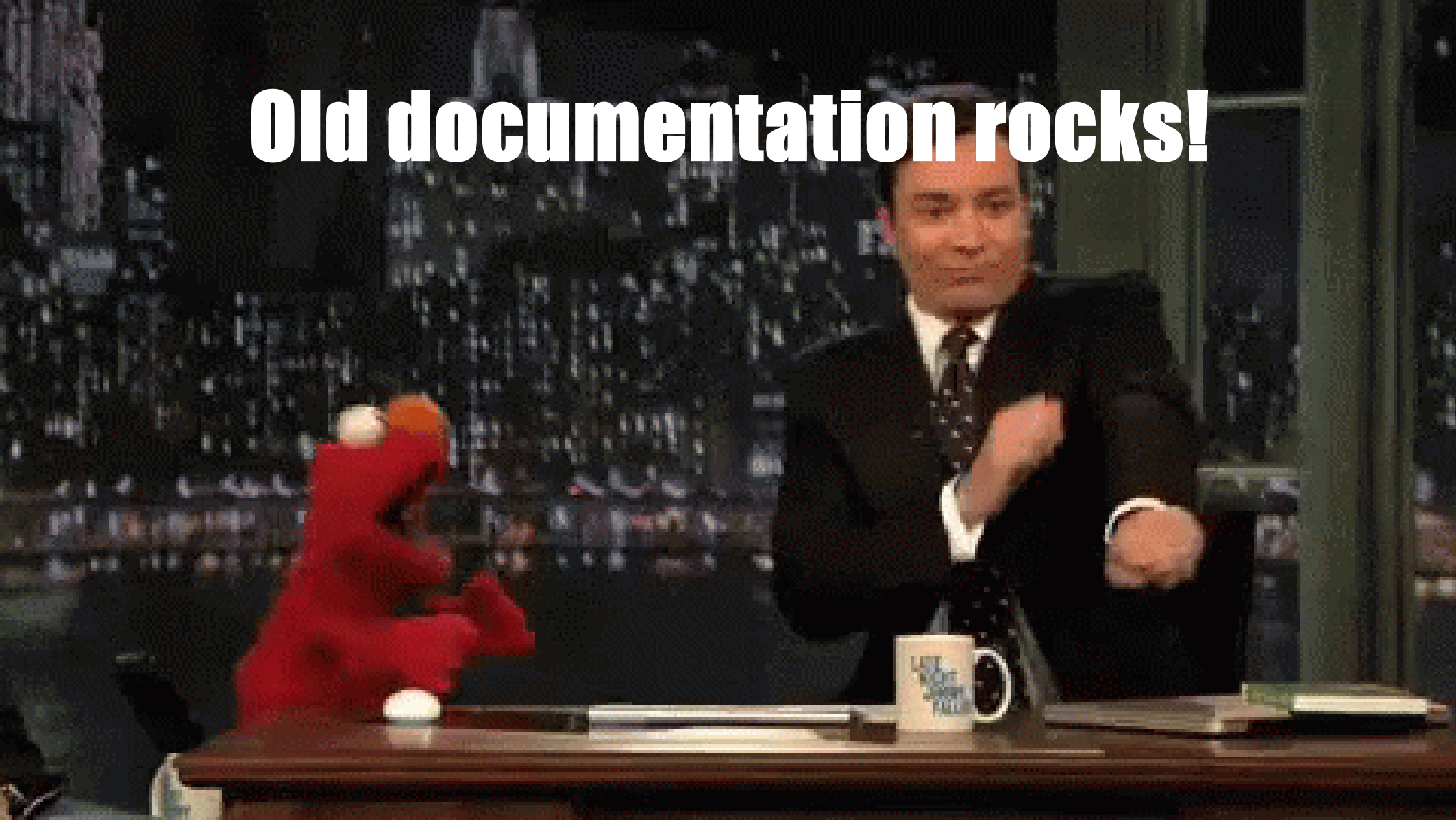
Why `interface`s? 💁‍♀️ (and not `type`s)

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

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

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

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

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


Old documentation rocks!

`type` aliases

type shortenings and manipulations

`interface`

object *shape* definitions

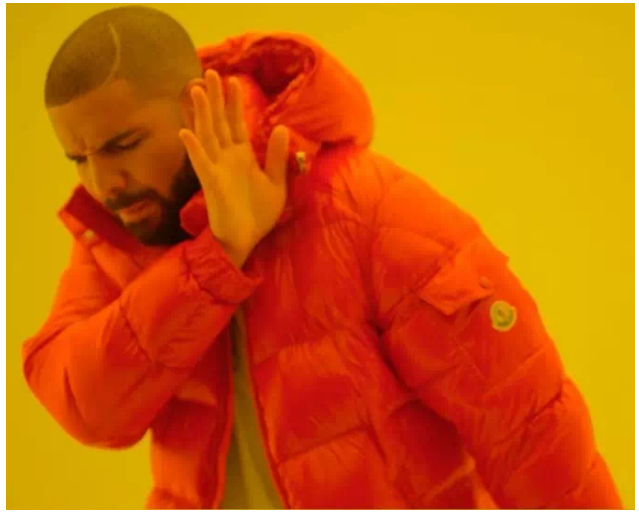
Models

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

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

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

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



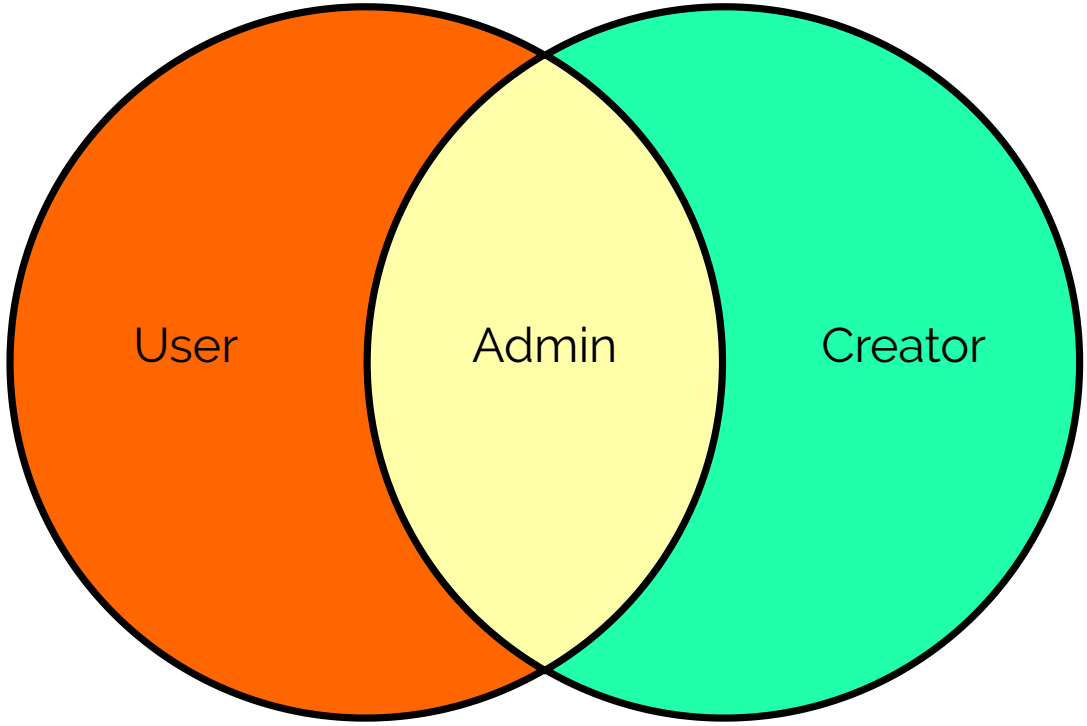
```
class Brand {
  name: string;
  constructor(name: string) {
    this.name = name;
  }
}
```


```
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;
```


User
Admin
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

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

```typescript
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
```

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

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

```typescript
interface UserWithId {
  id: string;
  name: string;
}
type UserKeys = keyof UserWithId; // 'id' | 'name'

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

```typescript
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'
```

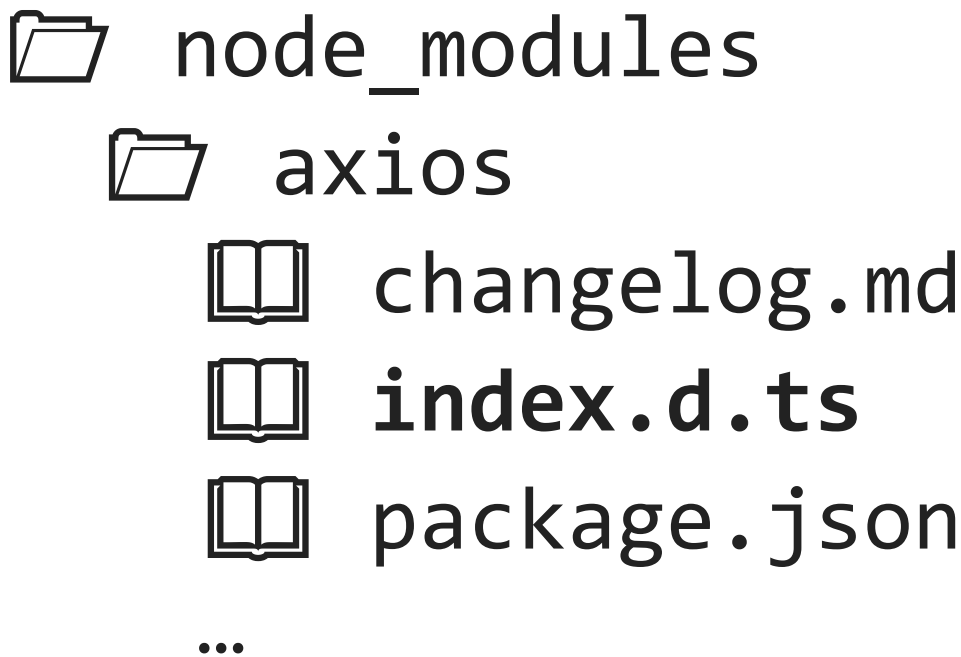
```typescript
interface IndexAction {
  kind: 'INDEX';
  index: number;
}
interface TitleAction {
  kind: 'TITLE';
  title: string;
}
type Action = IndexAction | TitleAction;
```

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

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

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


© Freeform

Where do all those definitions come from?

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



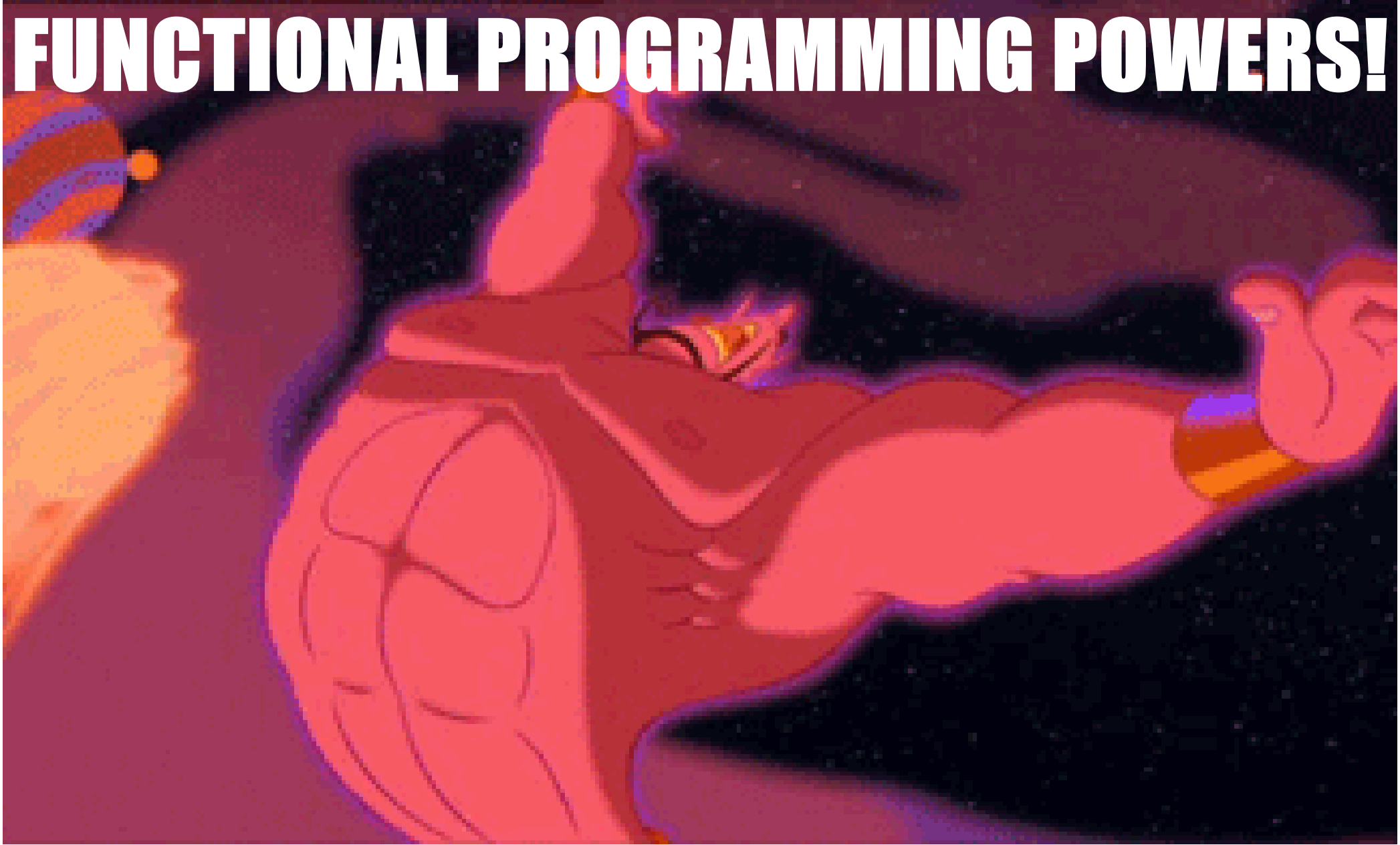

```
// packages.json
...
"typings": "types.d.ts"
```

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

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

Linting? 🤔

TSLint

`no-any` `no-empty-interface` `no-inferrable-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)
```

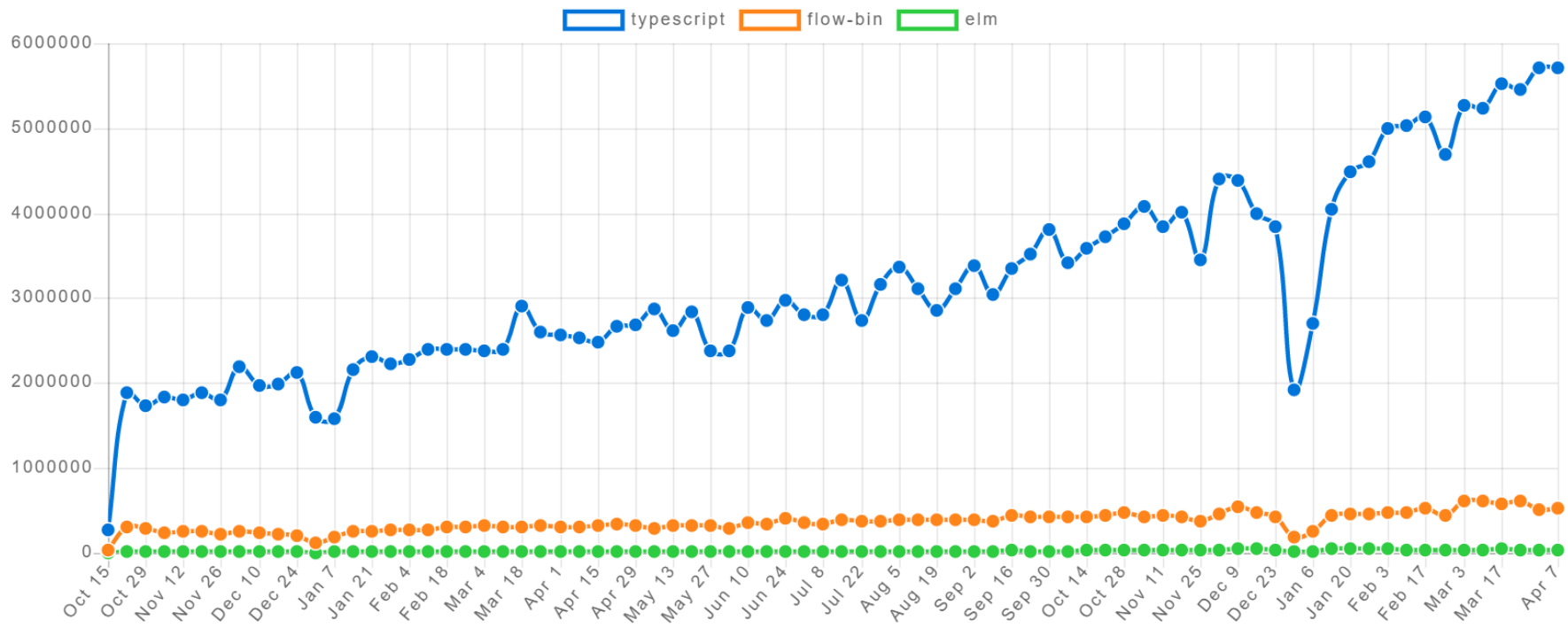
`--strict`

- `--noImplicitAny`
- `--noImplicitThis`
- `--alwaysStrict`
- `--strictBindCallApply`
- `--strictNullChecks`
- `--strictFunctionTypes`
- `--strictPropertyInitialization`

`--target`

`ES3`	`ES5`
`ES2015`	`ES2016`
`ES2017`	`ES2018`
`ESNext`	

typescriptlang.org/docs/handbook/compiler-options.html



Source: npmtrends.com

Bad reasons

to use TypeScript

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


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"

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

```
fu

}

co

);
```

```
+
```

Links

- Interface vs Type alias in TypeScript 2.7 @martin_hotell
  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
- Why use TypeScript, good and bad reasons @whereischarly
  itnext.io/why-use-typescript-good-and-bad-reasons-ccd807b292fb
- Italia JS Slack #typescript channel
  italiajs.slack.com/messages/CAHLHRZAQ/

That's all, folks!

maxart2501.github.io/typescript-talk/devromagna/

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