TYPESCRIPT

COME(FORSE)NON LO HAI MAI VISTO

italian agile movement

http://gianluca.carucci.org
gianlucacarucci5
https://www.facebook.com/caruccigianluca
https://www.linkedin.com/in/rucka

JOBS:

SOFTWARE ENGINEER & AGILE COACH@HYPERFAR INC

COMMUNITIES:

UGIDOTNET | SCALA MILANO MEETUP | MARKETERS

SLIDE & CODE:

HTTPS://GITHUB.COM/RUCKA/WORKINGSOFTWARE2019

OCTOBER 01, 2012

Anders Hejlsberg: Introducing TypeScript

Posted: Oct 01, 2012 at 8:45AM

By: Charles

(370) | 159 comments

Average: 5

TypeScript

Starts with JavaScript

All JavaScript code is TypeScript code, simply copy and paste

All JavaScript libraries work with TypeScript

Optional Static Types, Classes, Modules

Enable scalable application development and excellent tooling

Zero cost: Static types completely disappear at run-time

Ends with JavaScript

Compiles to idiomatic JavaScript

Runs in any browser or host, on any OS

WHAT TYPESCRIPT IS? 

IS

TYPESCRIPT == WEB + ❎

? 

IS

TYPESCRIPT == WEB + C#

? 🤔

NON-GOAL [1]

- EXACTLY MIMIC THE DESIGN OF EXISTING LANGUAGES. INSTEAD, USE THE BEHAVIOR OF JAVASCRIPT AND THE INTENTIONS OF PROGRAM AUTHORS AS A GUIDE FOR WHAT MAKES THE MOST SENSE IN THE LANGUAGE

[1] HTTPS://GITHUB.COM/MICROSOFT/TYPESCRIPT/WIKI/TYPESCRIPT-DESIGN-GOALS

NON-GOAL [1]

- EXACTLY MIMIC THE DESIGN OF EXISTING LANGUAGES. INSTEAD, USE THE BEHAVIOR OF JAVASCRIPT AND THE INTENTIONS OF PROGRAM AUTHORS AS A GUIDE FOR WHAT MAKES THE MOST SENSE IN THE LANGUAGE

[1] HTTPS://GITHUB.COM/MICROSOFT/TYPESCRIPT/WIKI/TYPESCRIPT-DESIGN-GOALS

GOALS [1]

- ALIGN WITH CURRENT AND FUTURE ECMASCRIPT PROPOSALS.
- USE A CONSISTENT. FULLY ERASABLE. STRUCTURAL TYPE SYSTEM.

[1] HTTPS://GITHUB.COM/MICROSOFT/TYPESCRIPT/WIKI/TYPESCRIPT-DESIGN-GOALS

GOALS [1]

- ALIGN WITH CURRENT AND FUTURE ECMASCRIPT PROPOSALS.
- USE A CONSISTENT. FULLY ERASABLE. STRUCTURAL TYPE SYSTEM.

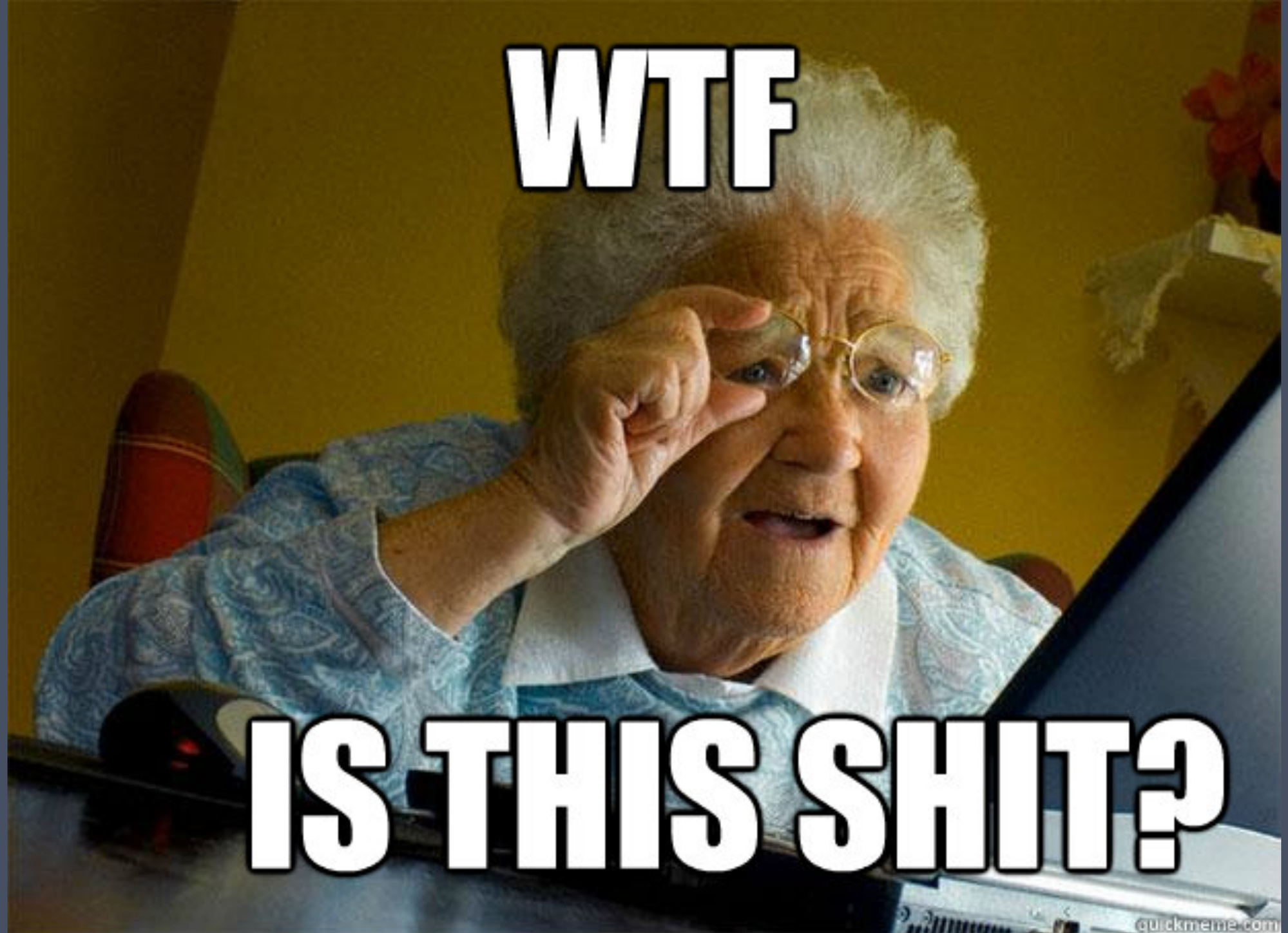
[1] HTTPS://GITHUB.COM/MICROSOFT/TYPESCRIPT/WIKI/TYPESCRIPT-DESIGN-GOALS

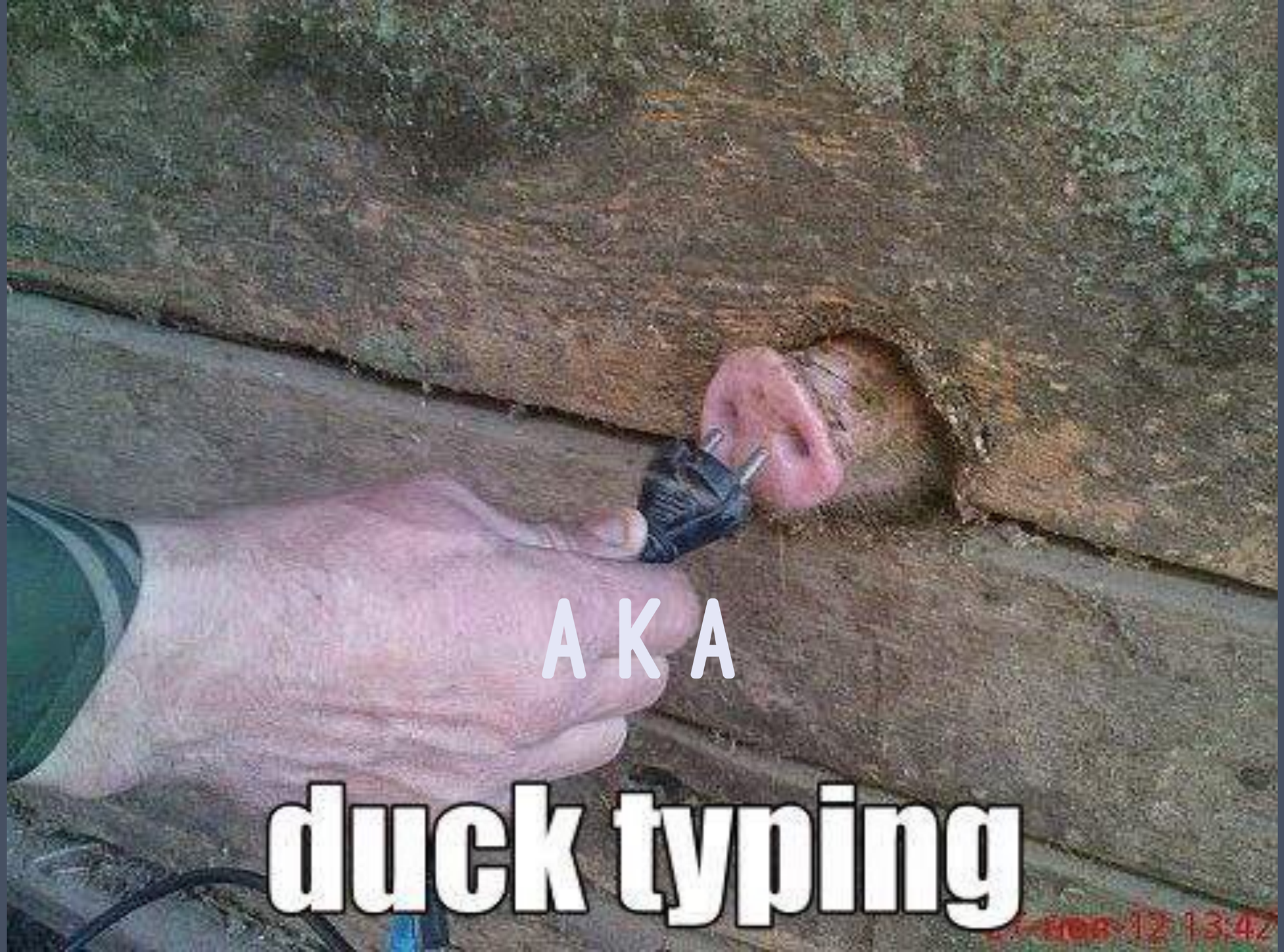
USE A CONSISTENT, FULLY ERASABLE, STRUCTURAL TYPE SYSTEM

USE A CONSISTENT, FULLY ERASABLE, STRUCTURAL TYPE SYSTEM

USE A CONSISTENT.
FULLY ERASABLE.
STRUCTURAL TYPE SYSTEM

STRUCTURAL TYPE SYSTEM


WTF
IS THIS SHIT?
quickmeme.com


A K A
duck typing

NOMINAL TYPING

```
class Foo {public string me;}
class Bar {public string me;}

Foo foo = new Foo(){me = "foo"};
Bar bar = foo; //<--
               //Error:
               //cannot implicit convert
               //type 'Foo' to 'Bar'
System.Console.WriteLine("I am "+ bar.me);
```

STRUCTURAL TYPING

```ts
interface Foo { me: string }
interface Bar { me: string }

const foo: Foo = { me: 'foo' }
const bar: Bar = foo

console.log(`I am ${bar.me}`); //I am foo
```

TYPESCRIPT + WEB + C#

DEMO

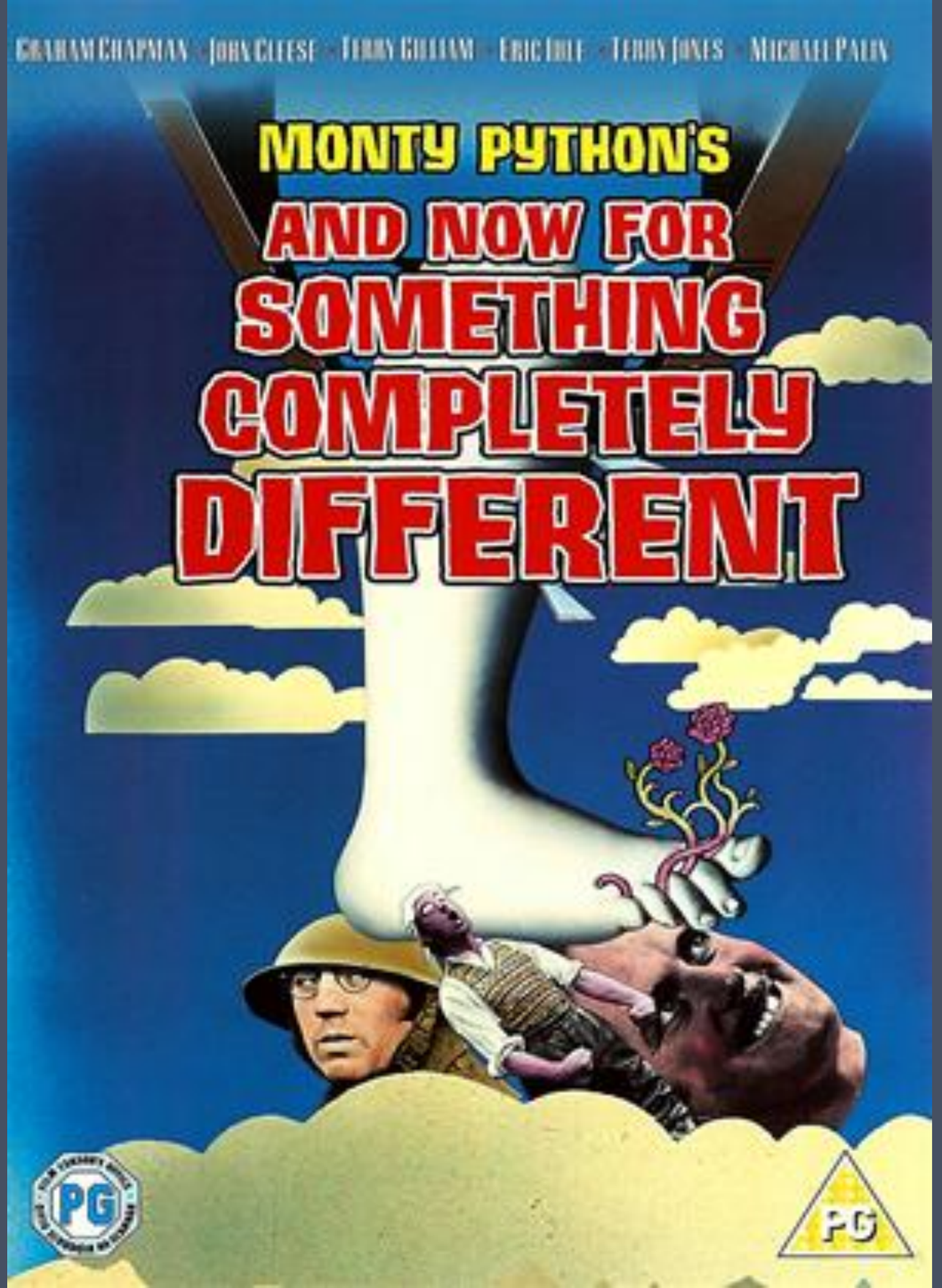
TYPESCRIPT: A GENTLE INTRODUCTION

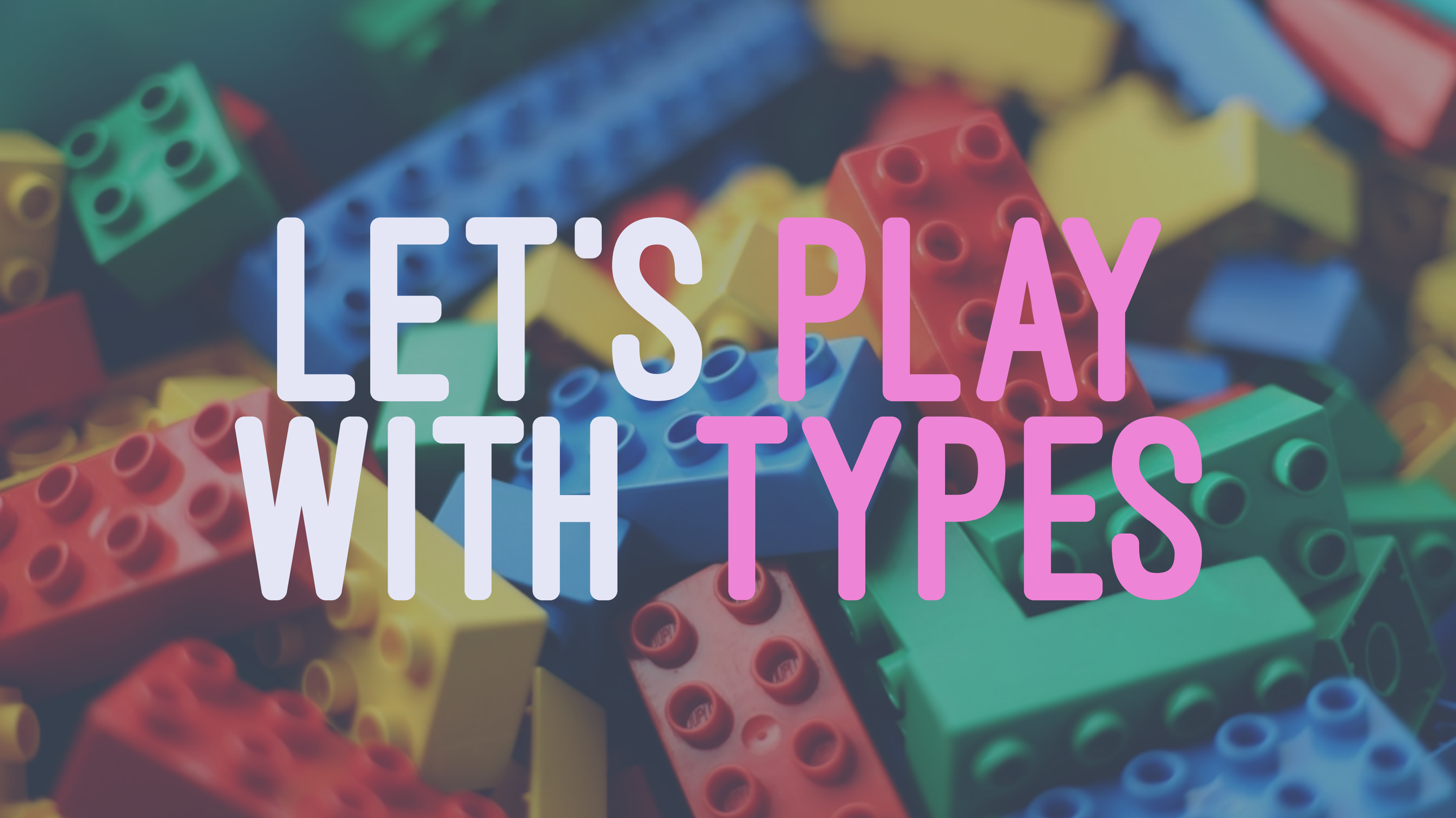
```ts
import { emailREx, urlREx, vatcodeREx } from '../_stuff/regex-collection'
import { valOf, checkedOf } from '../_stuff/dom'

const validateEmail = (email: string) => emailREx.test(email)
const validateVatCode = (vatcode: string) => vatcodeREx.test(vatcode)
const validateUrl = (url: string) => urlREx.test(url)

You, 12 days ago | 1 author (You)
class Customer {
  constructor(
    readonly name: string,
    readonly email: string,
    readonly website: string,
    readonly vatcode: string,
    readonly accepted: boolean
  ) {}
  isValid() {      You, 12 days ago • im  em  oop  lient v
    return this.validate().length === 0
  }
  validate() {
    let errors: string[] = []
    if (this.name === '') {
      errors.push('name must be filled')
    }
    if (this.vatcode === '') {
      errors.push('vat code must be filled')
    }
    if (this.email === '') {
      errors.push('email must be filled')
    }
    if (!this.accepted) {
      errors.push('terms must be accepted')
```

```
index.ts web/src/1-oop M
KLAB2019-3
  yargs-parser
  yn
  .yarn-integrity
  npm-packages-offline-cache
  references
    JoinTheExpert-WebDay-2016
      slides
  slides
  web
    src
      _stuff
      1-oop
        index.ts M
        2-vanillafp
          async.ts
          index.ts
          optional.ts
          validation.ts
        3-fp-ts
          index.ts
          liftA5.ts
        index.html
        index.ts
        style.css
```


GRAHAM CHAPMAN · JOHN CLEESE · TERRY GILLIAM · ERIC IDLE · TERRY JONES · MICHAEL PALIN
MONTY PYTHON'S
AND NOW FOR
SOMETHING
COMPLETELY
DIFFERENT
PG
PG


LET'S PLAY
WITH TYPES

GOALS [1]

> STATICALLY IDENTIFY CONSTRUCTS THAT ARE LIKELY TO BE ERRORS.

[1] HTTPS://GITHUB.COM/MICROSOFT/TYPESCRIPT/WIKI/TYPESCRIPT-DESIGN-GOALS

1. PARAMETRIC POLIMORPHISM[2]

```
interface Valid<A> { a: A }
interface Invalid<E> { errors: E[] }

function save<A>(data: Valid<A>): void {
    alert(`${JSON.stringfy(data.a)} saved!`);
}

function showErrors<E>(data: Invalid<E>): void {
    alert(`${JSON.stringfy(data.errors)} :(`);
}
```

[2] HTTPS://EN.WIKIPEDIA.ORG/WIKI/PARAMETRIC_POLYMORPHISM

2. UNION TYPES

```
type Validated<E, A> = Valid<A> | Invalid<E>

type CustomerError =
  | NameIsEmpty
  | NameIsTooShort
  | VatCodeIsEmpty
  | VatCodeNotValid
  | EmailIsEmpty
  | EmailIsNotValid
  | WebsiteUrlNotValid
  | TermsNotAccepted
```

3.1 DISCRIMINATED (TAGGED) UNION

```
interface NameIsTooShort { kind: 'NameIsTooShort', name: string }
interface VatCodeNotValid { kind: 'VatCodeNotValid', vatcode: string }
interface EmailIsNotValid { kind: 'EmailIsNotValid', email: string }
interface WebsiteUrlNotValid { kind: 'WebsiteUrlNotValid', url: string }

type CustomerError = NameIsTooShort | VatCodeNotValid | EmailIsNotValid | WebsiteUrlNotValid

function errorMessage(e: CustomerError): string {
    switch (e.kind) {
        case 'NameIsTooShort':
        return `Name must be at least 3 chars long, actual is ${e.name.length}`;
        case 'EmailIsNotValid':
        return `Email address '${e.email}' is not valid`;
        case 'VatCodeNotValid':
        return `VAT code '${e.vatcode}' is not valid`;
        case 'WebsiteUrlNotValid':
        return `Url '${e.url}' is not valid`;
    }
}
```

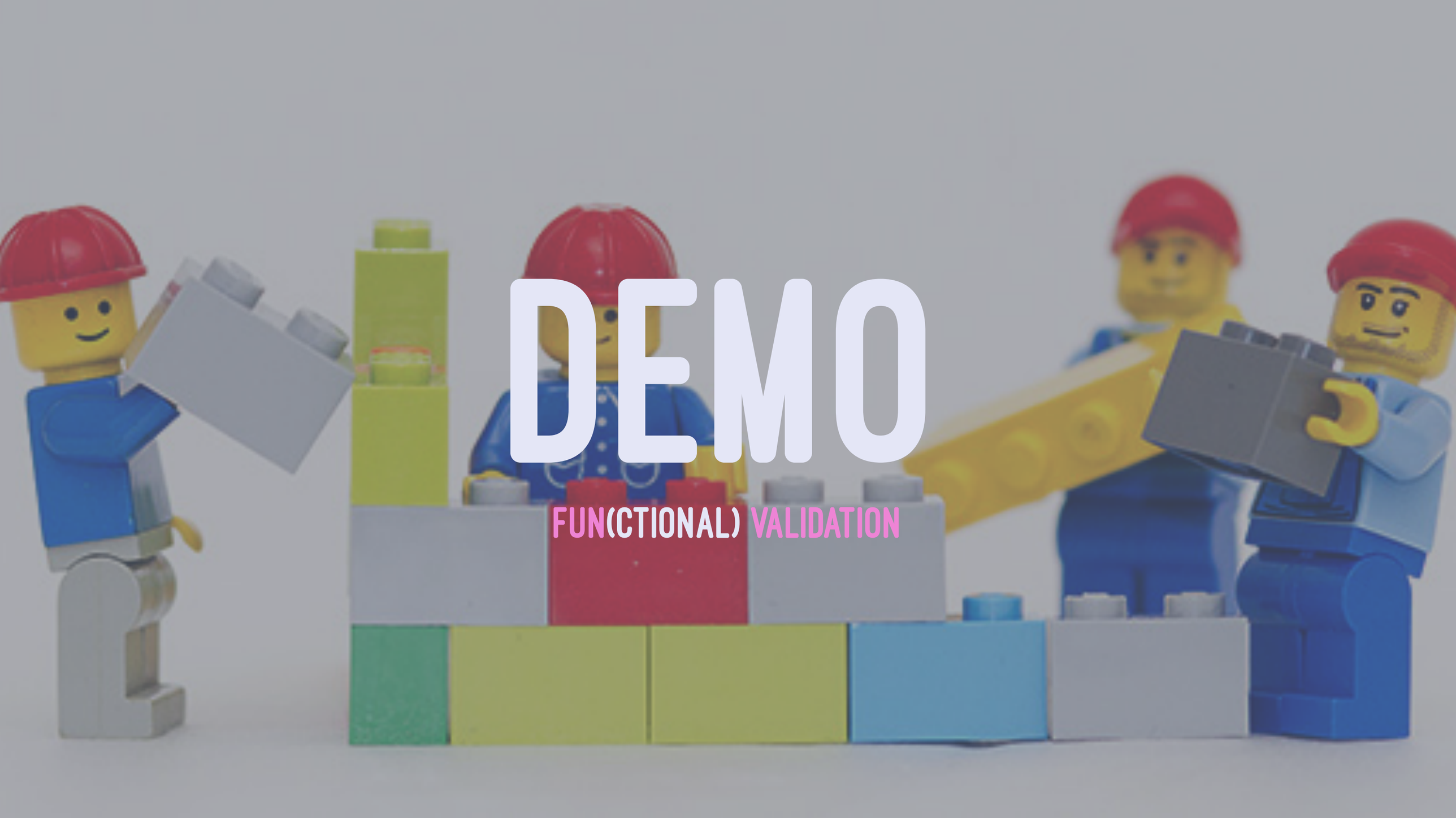
3.2 DISCRIMINATED (TAGGED) UNION

```
function isValid<A>(arg: any): arg is Valid<A> {
return arg.a !== undefined;
}

// here validatedCustomer is Validated<E, Customer>
if (isValid(validatedCustomer)) {
// here validatedCustomer is Valid<Customer>
return save(validatedCustomer);
} else {
// here validatedCustomer is Invalid<E>
return showErrors(validatedCustomer);
}
```

4. ITERSECTION TYPES

```
type A = {a: number}
type B = {b: number}
type I = A & B
let x: I = {a: 1, b: 2}; // OK
x = {a: 1, c: "three"}; // Error:
// Object literal may only specify known properties,
// and 'c' does not exist in type 'I'.
```



DEMO

FUN(CTIONAL) VALIDATION


USING FUNCTIONAL PROGRAMMING
YOU ARE STILL ALIVE
(MAYBE)

FUNCTIONAL API

FEATURES	VALIDATION	OPTIONAL (NULL)	TRY (EXCEPTION)	FUTURE (ASYNC)
X -> M[X]	lift	lift	lift	lift
(M[X1],M[X2]...M[Xn]) -> M[Y]	liftAn	liftAn	liftAn	liftAn
Projection	map	map	map	map
Filter	filter	filter	filter	filter


UNDER THE HOOD...

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Algebraic data type

From Wikipedia, the free encyclopedia

This article **needs additional citations for verification**. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed. *(October 2015)* *(Learn how and when to remove this template message)*

ALGEBRAIC DATA TYPE[3]

(AKA ADT)

In computer programming, more specifically functional programming and type theory, an **algebraic data type** is a kind of composite type, i.e., a type formed by combining other types.

Two common classes of algebraic types are product types (i.e., tuples and records) and sum types, also called tagged or disjoint unions or *variant types*.[1]

The values of a product type typically contain several values, called *fields*. All values of that type have the same combination of field types. The set of all possible values of a product type is the set-theoretic product, i.e., the Cartesian product, of the sets of all possible values of its field types.

The values of a sum type are typically grouped into several classes, called *variants*. A value of a variant type is usually created with a quasi-functional entity called a *constructor*. Each variant has its own constructor, which takes a specified number of arguments with specified types. The set of all possible values of a sum type is the set-theoretic sum, i.e., the disjoint union, of the sets of all possible values of its variants. Enumerated types are a special case of sum types in which the constructors take no arguments, as exactly one value is defined for each constructor.

Values of algebraic types are analyzed with pattern matching, which identifies a value by its constructor or field names and extracts the data it contains.

Algebraic data types were introduced in Hope, a small functional programming language developed in the 1970s at Edinburgh University.[2]

Contents [hide]

[3] HTTPS://EN.WIKIPEDIA.ORG/WIKI/ALGEBRAICDATATYPE

Functor

From Wikipedia, the free encyclopedia

This article is about the mathematical concept. For other uses, see Functor (disambiguation).

In mathematics, a **functor** is a type of mapping between categories which is applied in category theory. Functors can be thought of as homomorphisms between categories. In the category of small categories, functors can be thought of more generally as morphisms.

Functors were first considered in al[illegible] are associated to topological spaces, and algebraic homomorphisms are associated to [illegible] ma[illegible]s. [illegible] functors a[illegible] use[illegible] throughout modern mathematics to relate various categories. Thus, functors are generally applicable in areas within [illegible]ics that [illegible]ory can make abstract.

The word *functor* was borrowed by [illegible] fr[illegible] philosopher Rudolf C[illegible][3] who used the term in a linguistic context:[2] see function word.

FUNCTOR^4

Contents [hide]

^{4} HTTPS://EN.WIKIPEDIA.ORG/WIKI/FUNCTOR

Definition [edit]

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Monad (functional programming)

From Wikipedia, the free encyclopedia

For the object in category theory, see Monad (category theory).

In functional programming, **monads** are a way to build computer programs by joining simple components in robust ways. A monad encapsulates values of a particular data type, creating a new type associated with a specific computation. This computation follows a set of predicates called *monad laws*. The monad represents computations with a sequential structure: a monad defines what it means to chain operations together. This allows the programmer to build pipelines that process data in a series of steps (i.e. a series of *actions* applied to the data), in which each action is decorated with the additional processing rules provided by the monad.[1] A monad is defined by a *return* operator that creates values, and a bind operator used to link the actions in the pipeline.

Monads allow a programming style where programs are written by composing highly composable parts, combining in flexible ways the possible actions that can work on a particular type of data. As such, monads have been described as "programmable semicolons"; a semicolon is the operator used to chain together individual statements in many imperative programming languages, just as the expression implicitly adds extra code will be executed between the actions in the pipeline. Monads have also been explained with a physical metaphor as assembly lines, where a conveyor belt transports data between functional units that transform it one step at a time.[2] They can also be seen as a functional design pattern to build generic types.[3]

Purely functional programs can use monads to structure procedures that include sequenced operations like those found in structured programming.[4][5] Many common programming concepts can be described in terms of a monad structure without losing the beneficial property of referential transparency, including side effects such as input/output, variable assignment, exception handling, parsing, nondeterminism, concurrency, continuations, or domain-specific languages. This allows these concepts to be defined in a purely functional manner, without major extensions to the language's semantics. Languages like Haskell provide monads in the standard core, allowing programmers to reuse large parts of their formal definition and apply in many different libraries the same interfaces for combining functions.[6]

The name and concept comes from category theory, where monads are one particular kind of functor, a mapping between categories; however, in functional programming contexts the term *monad* is usually used with a meaning corresponding to that of the term *strong monad* in category theory.[7]

Contents [hide]

C# LINQ ❤️ MONAD[6] (LiftA*)

SelectMany == flatMap

```
var result =
    from r in rows
    from c in colums
    select "[" + r.Index + ", " + c.Index + "]";

var result = rows.SelectMany(r =>
    columns.SelectMany(c =>
        "[" + r.Index + ", " + c.Index + "]"));
```

[6] HTTP://MIKEHADLOW.BLOGSPOT.COM/2011/01/MONADS-IN-C-4-LINQ-LOVES-MONADS.HTML

Applicative functor

From Wikipedia, the free encyclopedia

In functional programming, specifically Haskell, an **applicative functor** is a structure that is like a monad (`return`, `fmap`, `join`) without `join`, or like a functor with `return`. Applicative functors are the programming equivalent of *strong lax monoidal functors* in category theory. In Haskell, applicative functors are implemented in the `Applicative` type class. Applicative functors were introduced in 2007 by Conor McBride and Ross Paterson in their paper *Functional Pearl: applicative programming with effects*.[1]

(A SORT OF)

APPLICATIVE[7]

Relationship to monad [edit]

Due to historical accident, applicative functors were not implemented as a subclass `Functor` nor as a superclass of `Monad`, but as a separate type class with no overlap. It turned out in practice, there was very little need for separation. So in 2014, it was proposed to make Applicative retroactively a subclass of Functor.[2]

See also [edit]

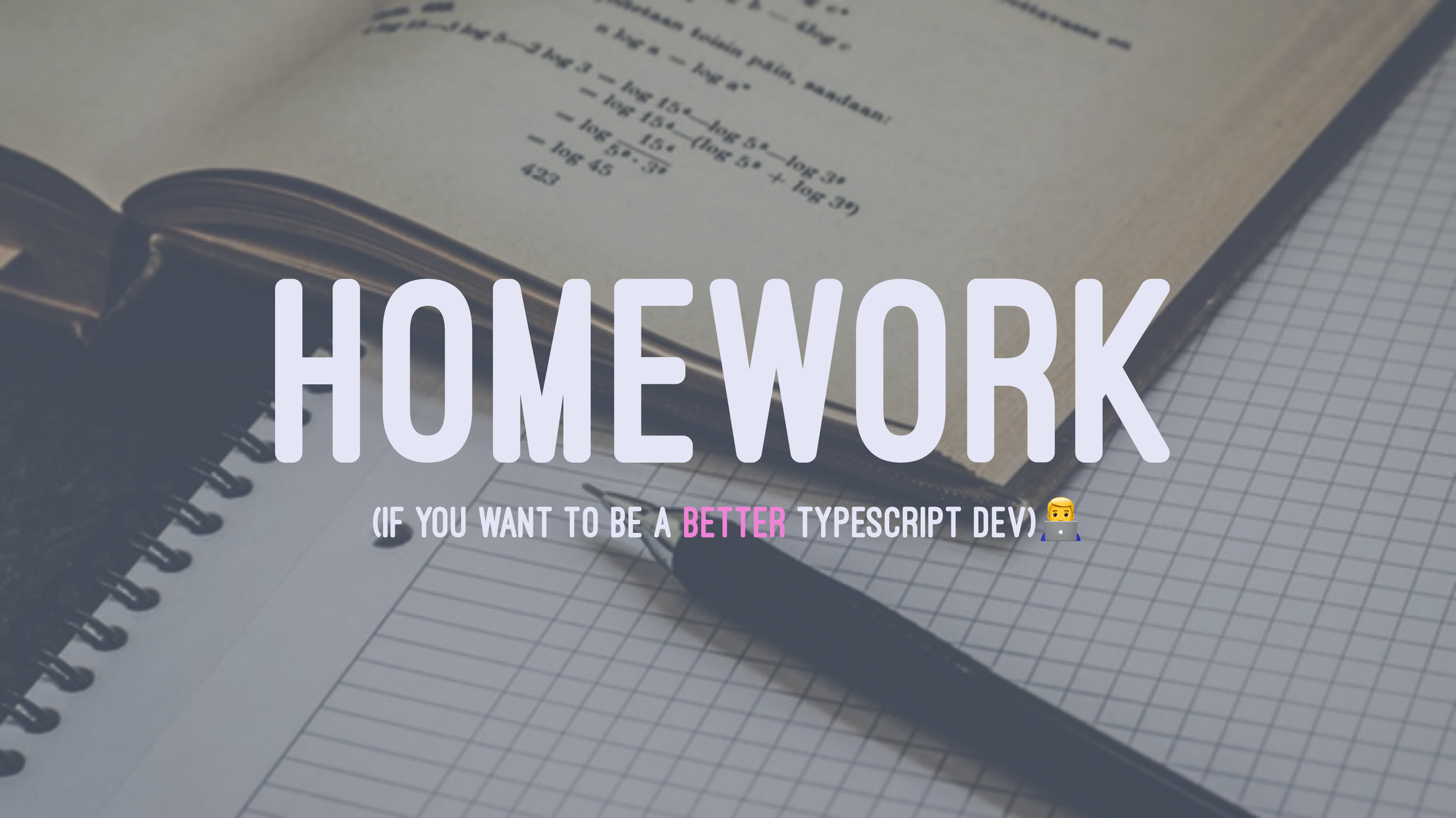
- Current definition of the Applicative class in Haskell

References [edit]

1. ^ Mcbride, Conor; Paterson, Ross (2008-01-01). "Applicative programming with effects". *Journal of Functional Programming*. **18** (01): 1–13. doi:10.1017/S0956796807006326. ISSN 1469-7653.
2. ^ "Functor-Applicative-Monad Proposal - HaskellWiki". *wiki.haskell.org*. Retrieved 2016-04-06.

This computer science article is a stub. You can help Wikipedia by expanding it.

7 HTTPS://EN.WIKIPEDIA.ORG/WIKI/APPLICATIVE_FUNCTOR


HOMEWORK
(IF YOU WANT TO BE A BETTER TYPESCRIPT DEV)👨‍💻

TypeScript

TypeScript 2.0 is now available!

September 22, 2016 by Daniel Rosenwasser // 91 Comments

[MANDATORY]

USE TYPESCRIPT >= 2.0

(CURRENT VERSION IS 3.4)

Today we're excited to announce the final release of TypeScript 2.0!

TypeScript 2.0 has been a great journey so far, with several contributions from the community paving the way. It brings several new features that enhance developer productivity, advance TypeScript's alignment with ECMAScript evolution, provides wide support for JavaScript libraries and tooling, and augments the language service that powers a host of editing experience across tools.

To get started, you can download TypeScript 2.0 for Visual Studio 2015 (which needs Update 3), grab it with NuGet, start using TypeScript 2.0 in Visual Studio Code, or install it with npm:

```
npm install -g typescript@2.0
```

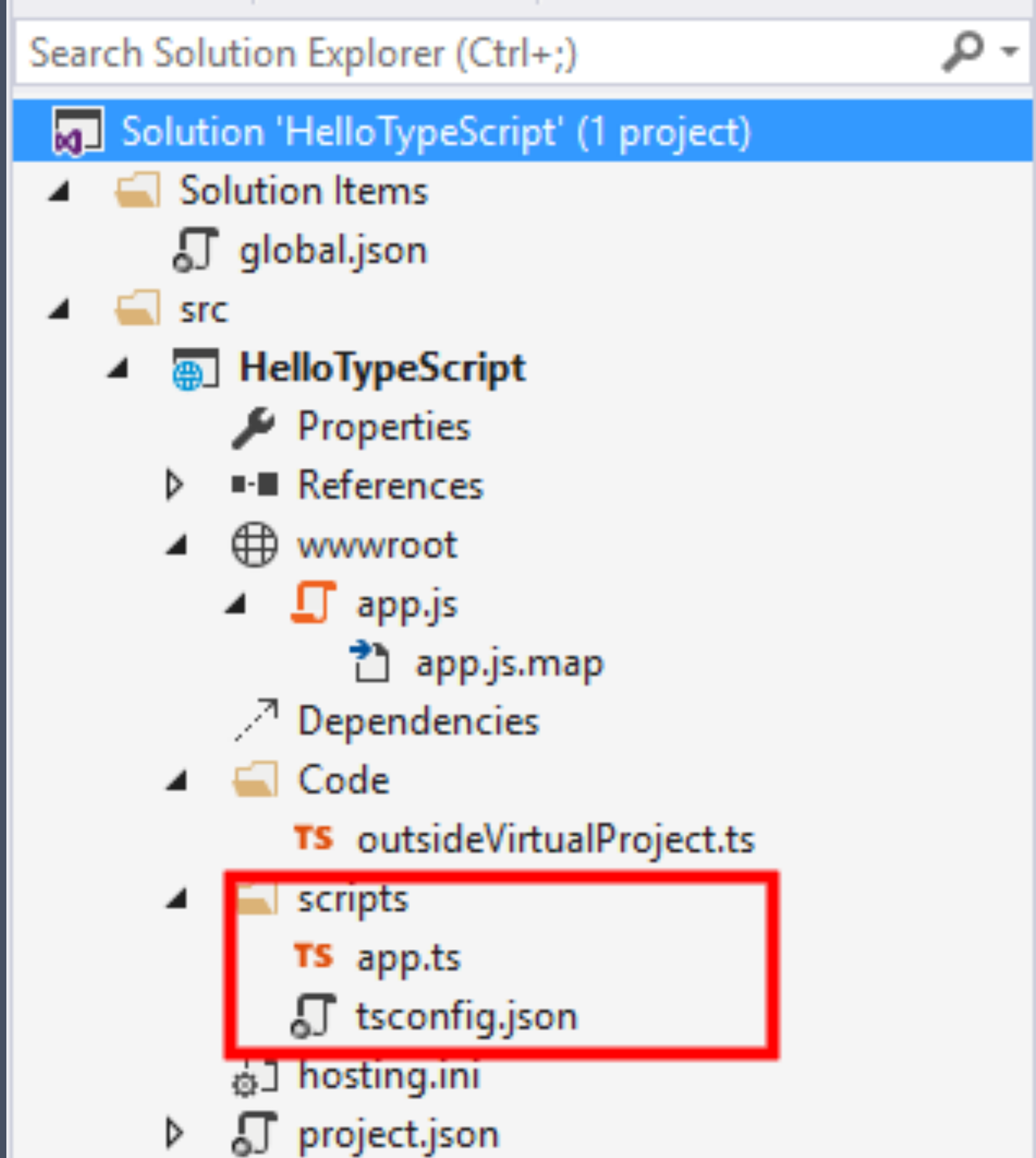
For Visual Studio "15" Preview users, TypeScript 2.0 will be included in the next Preview release.

The 2.0 Journey

A couple of years ago we set out on this journey to version 2.0. TypeScript 1.0 had successfully shown developers the potential of JavaScript when combined with static types. Compile-time error checking saved countless hours of bug hunting, and TypeScript's editor tools gave developers a huge productivity boost as they began building larger and larger JavaScript apps. However, to be a full superset of the most popular and widespread language in the world, TypeScript still had some growing to do.

[MANDATORY]

USE A CUSTOM TS.CONFIG



[MANDATORY]

ENABLE ALL COMPILER CHECKS

```
{
  "compilerOptions": {
    "strict": true,
    "noImplicitReturns": true,
    "noUnusedLocals": true,
    "noUnusedParameters": true,
    "noFallthroughCasesInSwitch": true,
    "noEmitOnError": true,
    "noImplicitAny": true,
    "noImplicitThis": true,
    "forceConsistentCasingInFileNames": true,
    "strictNullChecks": true,
    "alwaysStrict": true,
  }
}
```


TypeScript 2.0

Null- and undefined-aware types

TypeScript has two special types, Null and Undefined, that have the values `null` and
Previously it was not possible to explicitly name these types, but `null` and `undefin`
names regardless of type checking mode.

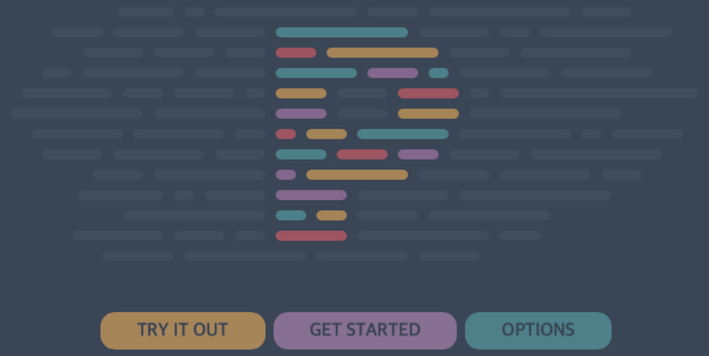
The type checker previously considered `null` and `undefined` assignable to anythi
`undefined` were valid values of *every* type and it wasn't possible to specifically exclu
possible to detect erroneous use of them).

`--strictNullChecks`

`--strictNullChecks` switches to a new strict null checking mode.

In strict null checking mode, the `null` and `undefined` values are *not* in the domai
assignable to themselves and `any` (the one exception being that `undefined` is also
`T` and `T | undefined` are considered synonymous in regular type checking mode
considered a subtype of any `T`), they are different types in strict type checking mode,
permits `undefined` values. The same is true for the relationship of `T` to `T | null`

Example



TRY IT OUT GET STARTED OPTIONS

What is Prettier?

- An opinionated code formatter
- Supports many languages
- Integrates with most editors
- Has few options

Why?

- You press save and code is formatted
- No need
- Saves yo
- And mor

rks with the Tools You Use

JavaScript
JSX
Flow
TypeScript
JSON

HTML
Vue
Angular

CSS
Less
SCSS
styled-components
styled-jsx

GraphQL
GraphQL Schemas

Markdown
CommonMark
GitHub-Flavored Markdown
MDX

YAML

Community Plugins
Elm (via elm-format)
Java
PHP
PostgreSQL
Ruby
Swift
TOML

```
lo", greeted = '"World"', silent = false, onMouseOver,}

ender
) * 1E+7).toString().replace(/\.\d+/ig, "")

 title={`You are visitor number ${ num }`} onMouseOver=

.toUpperCase() + greeting.slice(1).toLowerCase() }</str
<span style={{color: '\grey'}}>", "</span> }
```

```
function HelloWorld({
  greeting = "hello",
  greeted = '"World"',
  silent = false,
  onMouseOver
}) {
  if (!greeting) {
    return null;
  }

  // TODO: Don't use random in render
  let num = Math.floor(Math.random() * 1e7)
    .toString()
    .replace(/\.\d+/gi, "");

  return (
    <div
      className="HelloWorld"
      title={`You are visitor number ${num}`}
      onMouseOver={onMouseOver}
    >
      <strong>
        {greeting.slice(0, 1).toUpperCase() + greet
      </strong>
      {greeting.endsWith(",") ? (
        " "
      ) : (
        <span style={{ color: "grey" }}>", "</span>
      )}
      <em>{greeted}</em>
      {silent ? "." : "!"}
    </div>
  );
}

```

[SUGGESTION]

USE PRETTIER[8]

[8] HTTPS://PRETTIER.IO

Modules

> **A note about terminology:** It's important to note that in TypeScript 1.5, the nomenclature has changed. "Internal modules" are now "namespaces". "External modules" are now simply "modules", as to align with ECMAScript 2015's terminology, (namely that `module X {` is equivalent to the now-preferred `namespace X {`).

Introduction

PREFER MODULES OVER NAMESPACES

Starting with the ECMAScript 2015, JavaScript has a concept of modules. TypeScript shares this concept.

Modules are executed within their own scope, not in the global scope; this means that variables, functions, classes, etc. declared in a module are not visible outside the module unless they are explicitly exported using one of the `export` forms. Conversely, to consume a variable, function, class, interface, etc. exported from a different module, it has to be imported using one of the `import` forms.

Modules are declarative; the relationships between modules are specified in terms of imports and exports at the file level.

Modules import one another using a module loader. At runtime the module loader is responsible for locating and executing all dependencies of a module before executing it. Well-known modules loaders used in JavaScript are the CommonJS module loader for Node.js and require.js for Web applications.

In TypeScript, just as in ECMAScript 2015, any file containing a top-level `import` or `export` is considered a module.

Search fp-ts

Typed functional programming in Ty

fp-ts provides developers with popular patterns and re
from typed functional languages in TypeScript.

Get started | Take the tutorial

Installation

To install the stable version:

github.com

gcanti/fp-ts: Functional p

Pull requests Issues Marketplace Explore

gcanti / fp-ts

Watch 79 | Unstar 2,151

Code | Issues 35 | Pull requests 6 | Insights

Functional programming in TypeScript https://gcanti.github.io/fp-ts/

typescript | functional-programming | algebraic-data-types

1,412 commits | 7 branches | 82 releases | 48 contributors

Branch: master | New pull request | Create new file | Upload files | Clone | Find File

gcanti version 1.18.1 — Latest commit c458

.github	Docs: add PULL_REQUEST_TEMPLATE
.vscode	Docs: format comments with `rewrap` VS Code extension
docs	fix Record.reduce example
dtslint	Defect Fix: remove `reverse` (mutable) from `NonEmptyArr...
examples	Move all data structures in the `examples` folder to fp-ts-c...
fantas-eel-and-specifi...	Polish: remove tsconfig from fantas-eel-and-specification
perf	optimize function.concat
src	Use explicit concat function for getEndomorphismMonoid
test	New Feature: add absurd function, closes #847
tutorials	Polish: make Type<URI1, A> not assignable to Type<URI2, ...
.editorconfig	FR: migrate to curried // update project
.gitignore	Add es6 module step to build for tree-shaking support
.prettierignore	Docs: fix docs
.prettierrc	Docs: [ramda] add links
.travis.yml	upgrade node version
CHANGELOG.md	version 1.18.1
LICENSE	update LICENSE
README.md	Document fp-ts cjs & es6 import options
code2comment.html	remove old docs generator
jest.config.js	add property test
package-lock.json	version 1.18.1
package.json	version 1.18.1
tsconfig.es6.json	Don't set `target: es6` in tsconfig.es6.json
tsconfig.json	version 1.14.1
tsconfig.tslint.json	Polish: enable tslint on tests
tslint.json	add no-inferred-empty-object-type lint rule

USE FP-TS

THE RIGHT TOOL FOR THE RIGHT JOB


QUESTIONS?


Mi piace
Segui
Condividi
Gianluca Carucci
22 aprile alle ore 12:02 ·
[COME IL LEAN THINKING PUO' FAR DIVENTARE LA TUA AZIENDA VINCENTE🚀🚀🚀]
Da cosa dipende il successo di una azienda?
Fatturato? Seguito sui (social)media?
Voglio condividere con te questo pensiero.
🎤 "Non confondete mai il valore con il prezzo. Quando un consumatore compra un prodotto, lo fa perché quel prodotto ha un certo valore per lui. Ma il costo per produrlo è alto e così voi aumentate il prezzo! Non scegliete mai questa strada facile. Se aumentate il prezzo ma il valore r...
Continua a leggere
[COS'E' SCRUM E PERCHE' POTREBBE ESSERTI UTILE⚡⚡⚡]
📌 Scrum ti serve?
Se non hai mai sentito questo nome, sappi che non sto parlando di u
trattamento per il viso.
Magari ti fa - giustamente - pensare al rugby e ti chiedi perché te ne
parlando in questo contesto.🏉...
Continua a leggere
SE HAI TROVATO QUESTO TALK INTERESSANTE,
LEGGI ALTRI CONTENUTI SU:
Gianluca Carucci
31 dicembre 2018 ·
RENDERE UTILI LE RIUNIONI🚀🚀🚀
Questa riunione è stata una perdita
PER GESTIRE I PERIODI DI LAVORO INTENSO DE
venerdì.
se lo ricorda bene anche il mio team
SEBASTIENLOEBRACING


Thank You!

workingsoftware conference 2019