



TYPESCRIPT

COME(FORSE)NON LO HAI MAI VISTO

<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](https://github.com/rucka/workingsoftware2019)



1 OCTOBER 2012

Anders Hejlsberg: Introducing TypeScript

Posted: Oct 01, 2012 at 8:45AM

By: Charles

★★★★★ (370) | 159 comments

Average: 5

TypeScript

Charts with JavaScript

A 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-GOAL¹

- 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](https://github.com/microsoft/TypeScript/wiki/TypeScript-Design-Goals)

NON-GOAL¹

- > 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](https://github.com/microsoft/TypeScript/wiki/TypeScript-Design-Goals)

GOALS¹

- > 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](https://github.com/microsoft/TypeScript/wiki/TypeScript-Design-Goals)

GOALS¹

- > 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](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**

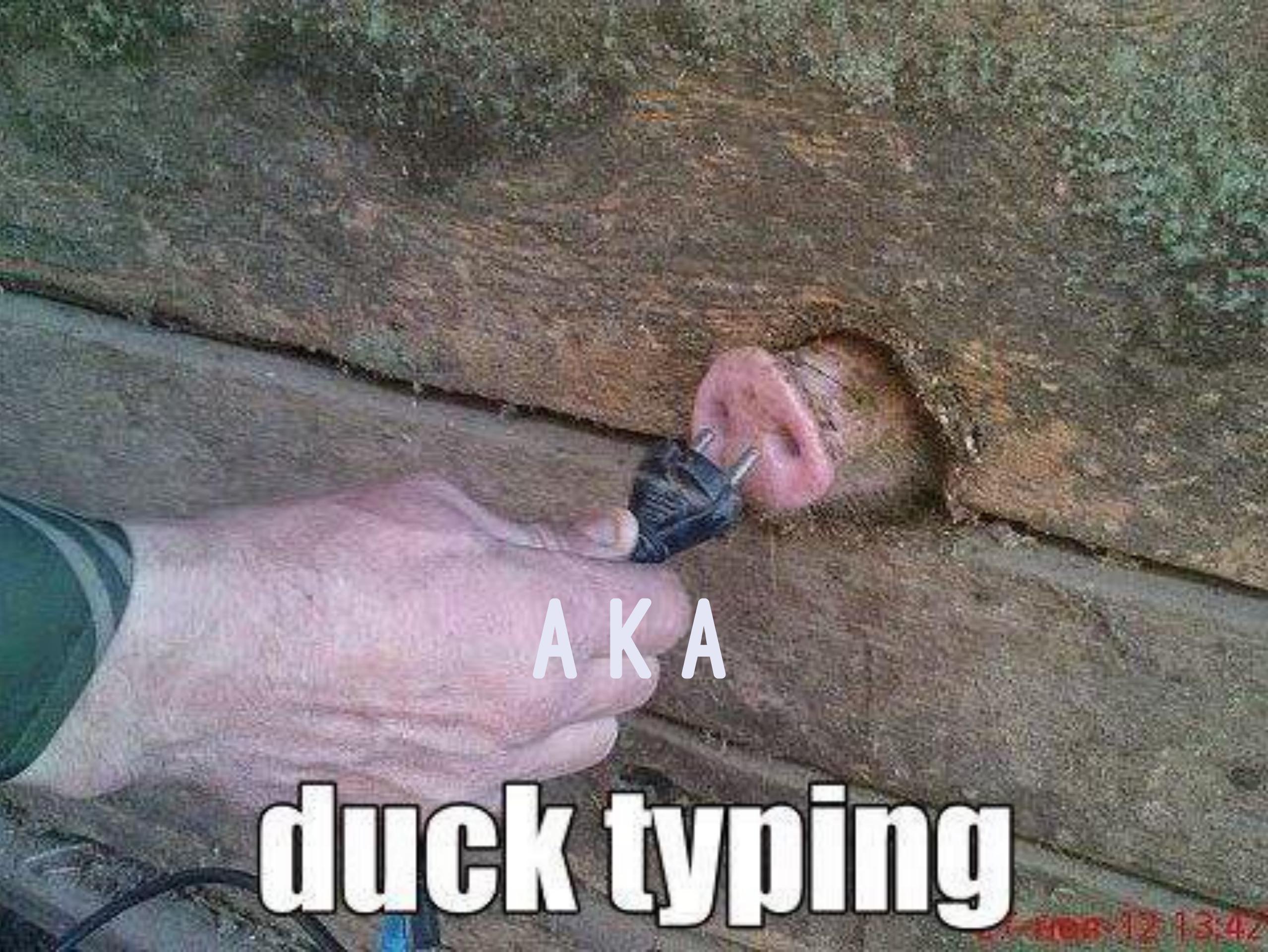


STRUCTURAL TYPE SYSTEM



WTF

IS THIS SHIT?



A K A
duck typing

300-105-1342

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

```
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
```



TYPERIGHT
TOUR
WLD C#

```
1 import { emailREx, urlREx, vatcodeREx } from '../_stuff/regex-collection'
2 import { valOf, checkedOf } from '../_stuff/dom'
3
4 const validateEmail = (email: string) => emailREx.test(email)
5 const validateVatCode = (vatcode: string) => vatcodeREx.test(vatcode)
6 const validateUrl = (url: string) => urlREx.test(url)
7
8 You, 12 days ago | 1 author (You)
9 class Customer {
10   constructor(
11     readonly name: string,
12     readonly email: string,
13     readonly website: string,
14     readonly vatcode: string,
15     readonly accepted: boolean
16   ) {}
17   isValid(): boolean {
18     return this.validate().length === 0
19   }
20   validate(): string[] {
21     let errors: string[] = []
22     if (this.name === '') {
23       errors.push('name must be filled')
24     }
25     if (this.vatcode === '') {
26       errors.push('vat code must be filled')
27     }
28     if (this.email === '') {
29       errors.push('email must be filled')
30     }
31     if (!this.accepted) {
32       errors.push('terms must be accepted')
33     }
34     return errors
35   }
36 }
```

DEMO

TYPESCRIPT: A GENTLE INTRODUCTION

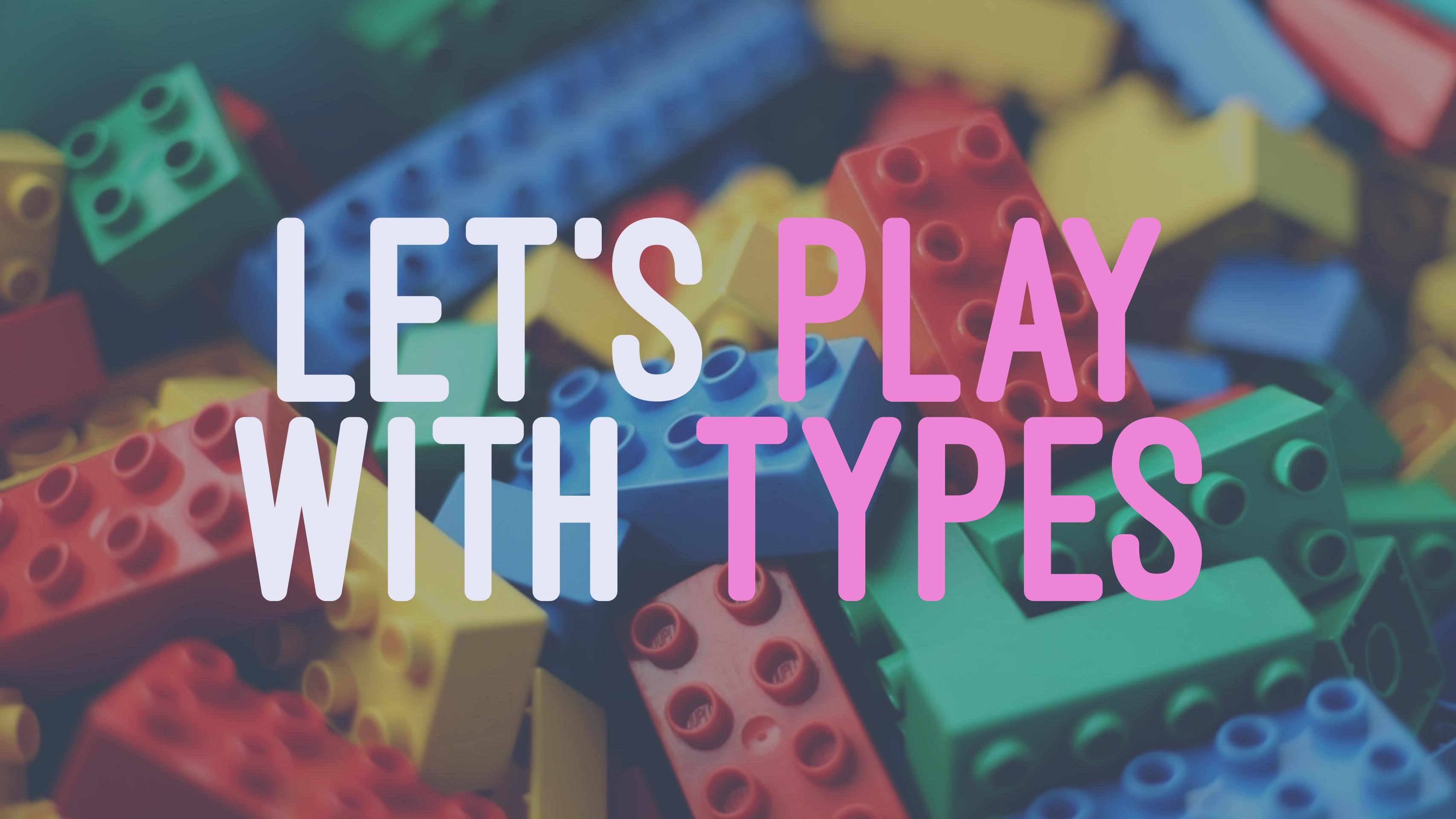
The screenshot shows a file system view on the right side of a code editor. The root folder is 'KLAB2019-3'. Inside it are several subfolders and files:

- yargs-parser
- yn
- .yarn-integrity
- npm-packages-offline-cache
- references
- JoinTheExpert-WebDay-2016
 - slides
- web
- src
 - _stuff
 - 1-oop
 - index.ts
- 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 CLESE · TERRY GILLIAM · ERIC IDLE · TERRY JONES · MICHAEL PALIN

MONTY PYTHON'S
**AND NOW FOR
SOMETHING
COMPLETELY
DIFFERENT**





LET'S PLAY
WITH TYPES

GOALS¹

- > STATICALLY IDENTIFY CONSTRUCTS THAT ARE LIKELY TO BE ERRORS.

¹ [HTTPS://GITHUB.COM/MICROSOFT/TYPESCRIPT/WIKI/TYPESCRIPT-DESIGN-GOALS](https://github.com/microsoft/TypeScript/wiki/TypeScript-Design-Goals)

1. PARAMETRIC POLIMORPHISM²

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

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

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

² [HTTPS://EN.WIKIPEDIA.ORG/WIKI/PARAMETRIC_POLYMORPHISM](https://en.wikipedia.org/wiki/Parametric_polyorphism)

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`;
  }
}
```

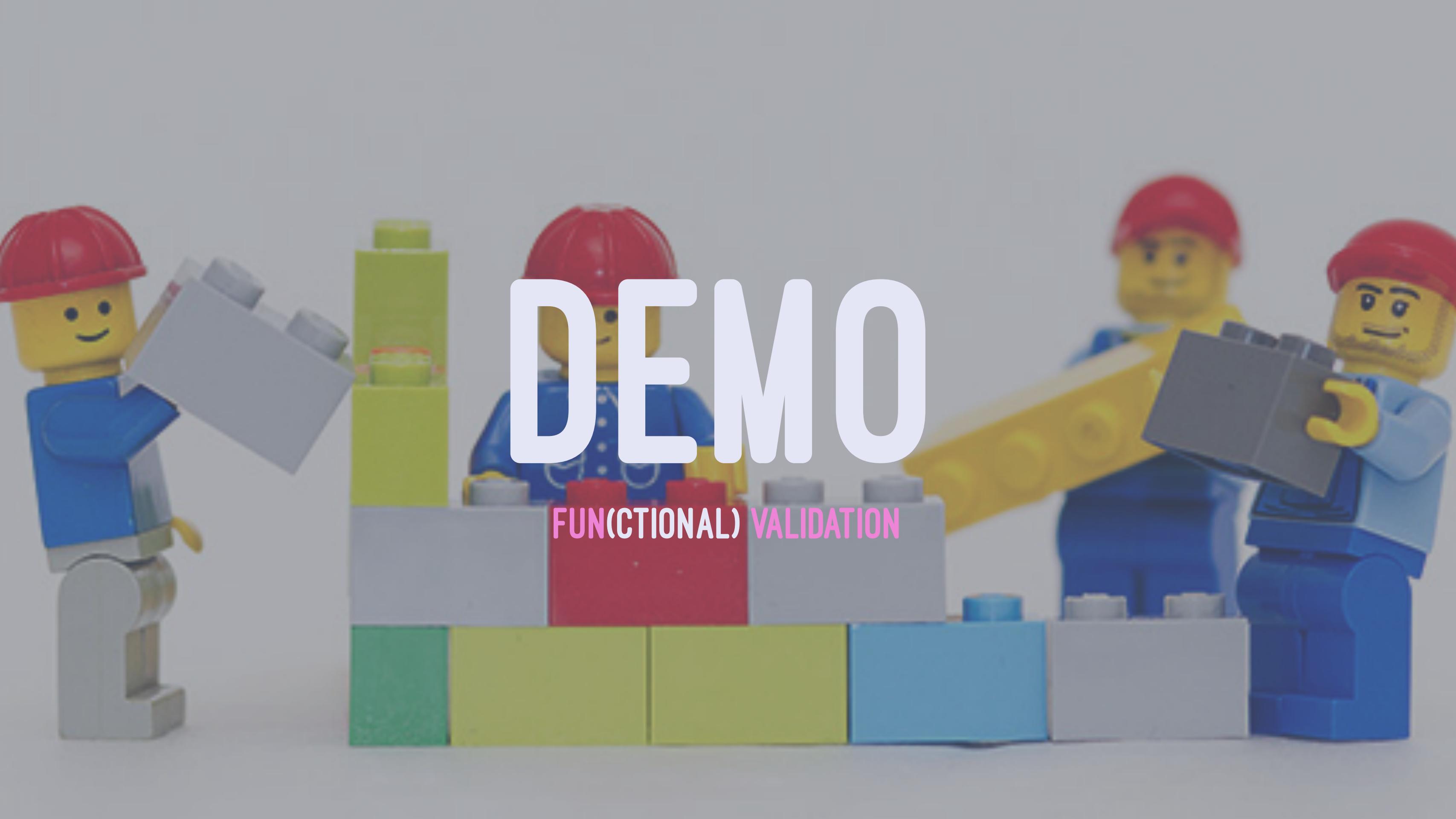
3.2 DISCRIMINATED (TAGGED) UNION

```
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. INTERSECTION 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

A photograph of a row of white front-loading washing machines in a public laundry facility. A person's arm, wearing a blue long-sleeved shirt and a brown leather bracelet, reaches out from the right side of the frame towards the center. In the foreground, there is a metal laundry cart with wheels, partially filled with laundry. The ceiling has recessed lighting, and there are doors and a sign in the background.

USING FUNCTIONAL PROGRAMMING
YOU ARE STILL ALIVE
(MAYBE)



FUNCTIONAL API

| FEATURES | VALIDATION | OPTIONAL (NULL) | TRY (EXCEPTION) | FUTURE (ASYNC) |
|--|------------|-----------------|-----------------|----------------|
| $X \rightarrow M[X]$ | lift | lift | lift | lift |
| $(M[X_1], M[X_2] \dots M[X_n]) \rightarrow M[Y]$ | liftAn | liftAn | liftAn | liftAn |
| Projection | map | map | map | map |
| Filter | filter | filter | filter | filter |

A hooded figure, possibly a monk or hermit, is shown from the chest up. They are wearing a dark hood and a red robe with a white cross on the chest. A bright, glowing orange and yellow orb, resembling a small fire or a celestial body, is held in their hands, emitting a strong light that illuminates their face and the surrounding area. The background is dark and out of focus, suggesting a forest or a cave.

UNDER THE HOOD...



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 TYPE³

In computer programming, more specifically in type theory, an **algebraic data type** is a **sum type** or **product 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]

- 1 Examples
- 2 Explanation
- 3 Theory
- 4 Programming languages with algebraic data types
- 5 See also
- 6 References

³ [HTTPS://EN.WIKIPEDIA.ORG/WIKI/ALGEBRAICDATA TYPE](https://en.wikipedia.org/wiki/Algebraic_data_type)

[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](#)[Article](#) [Talk](#)[Read](#) [Edit](#) [View history](#)[Search Wikipedia](#)

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 algebraic topology, where they generalize objects like the fundamental group, which are associated to topological spaces, and algebraic homomorphisms are associated to continuous maps. Nowadays, functors are used throughout modern mathematics to relate various categories. Thus, functors are generally applicable in areas within mathematics that are not directly related by the theory of categories, such as algebraic topology and algebraic geometry.

The word *functor* was borrowed by mathematicians from the philosopher Rudolf Carnap[1] who used the term in a linguistic context:^[2] see [function word](#).

Contents [hide]

- 1 Definition
 - 1.1 Covariance and contravariance
 - 1.2 Opposite functor
 - 1.3 Bifunctors and multifunctors
- 2 Examples
- 3 Properties
- 4 Relation to other categorical concepts
- 5 Computer implementations
- 6 See also
- 7 Notes
- 8 References
- 9 External links

⁴ [HTTPS://EN.WIKIPEDIA.ORG/WIKI/FUNCTOR](https://en.wikipedia.org/wiki/Functor)

Definition [\[edit\]](#)



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.

(A SORT OF)
MONAD⁵

Monads allow a [programming style](#) where programs are written by using together 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 "programmatic semicolons"; a semicolon is the operator used to chain together individual statements in many [imperative programming languages](#).^[1] Plus, the entire code block after the 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 [\[hide\]](#)

[HTTPS://EN.WIKIPEDIA.ORG/WIKI/MONAD\(FUNCTIONAL PROGRAMMING\)](https://en.wikipedia.org/wiki/Monad_(functional_programming))

- 1 Overview
- 2 History
- 3 Motivating examples

C# LINQ ❤ MONAD⁶ (LiftA*)

SelectMany == flatMap

```
var result =  
    from r in rows  
    from c in columns  
    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](http://MIKEHADLOW.BLOGSPOT.COM/2011/01/MONADS-IN-C-4-LINQ-LOVES-MONADS.HTML)

[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](#)[Article](#) [Talk](#)[Read](#) [Edit](#) [View history](#)[Search Wikipedia](#)

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)

Relationship to monads

Due to historical accident, applicative functors were not implemented as a subclass of `Functor`, and does a superclass of `Monad`, but as a separate type class with no overlap. It turned out, in practice, there was very little demand for separation so in 2014, it was proposed to make Applicative retroactively a subclass of `Functor`.^[2]

APPLICATIVE⁷

See also

- [Current definition of the Applicative class in Haskell](#)

References

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](http://wiki.haskell.org/Functor-Applicative-Monad_Proposal). Retrieved 2016-04-06.



This computer science article is about the concept. You may be looking for the term in computing.

[HTTPS://EN.WIKIPEDIA.ORG/WIKI/APPLICATIVE_FUNCTOR](https://en.wikipedia.org/w/index.php?title=Applicative_functor&oldid=711111117)

HOMEWORK

(IF YOU WANT TO BE A BETTER TYPESCRIPT DEV) 

| | | | | | | | |
|--------------------|---------------|----------------------------------|-----------|------|----------------------|------------------|--|
| Executive Bloggers | Visual Studio | Application Lifecycle Management | Languages | .NET | Platform Development | Data Development | |
|--------------------|---------------|----------------------------------|-----------|------|----------------------|------------------|--|

TypeScript

TypeScript 2.0 is now available!

September 22, 2016 by Daniel Rosenwasser // 91 Comments



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

TypeScript 2.0 has been a great journey for the team, with several contributions from the community and partners along the way. It brings several new features that enhance developer productivity, advances TypeScript's alignment with ECMAScript evolution, provides wide support for JavaScript libraries and tools, and augments the language services that power a rich developer 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.

TypeScript 1.1 brought a new, completely rewritten compiler that delivered a 4x performance boost. This new compiler core allowed more



[Download Visual Studio](#) →

[Download Visual Studio Code](#) →

[Search MSDN with Bing](#)



Search this blog Search all blogs

Share This Post



Recent Posts

TypeScript 2.0 is now available! September 22, 2016

Announcing TypeScript 2.0 RC August 30, 2016

Announcing TypeScript 2.0 Beta July 11, 2016

The Future of Declaration Files June 15, 2016



Back to top

Live Now on [Developer Tools Blogs](#)

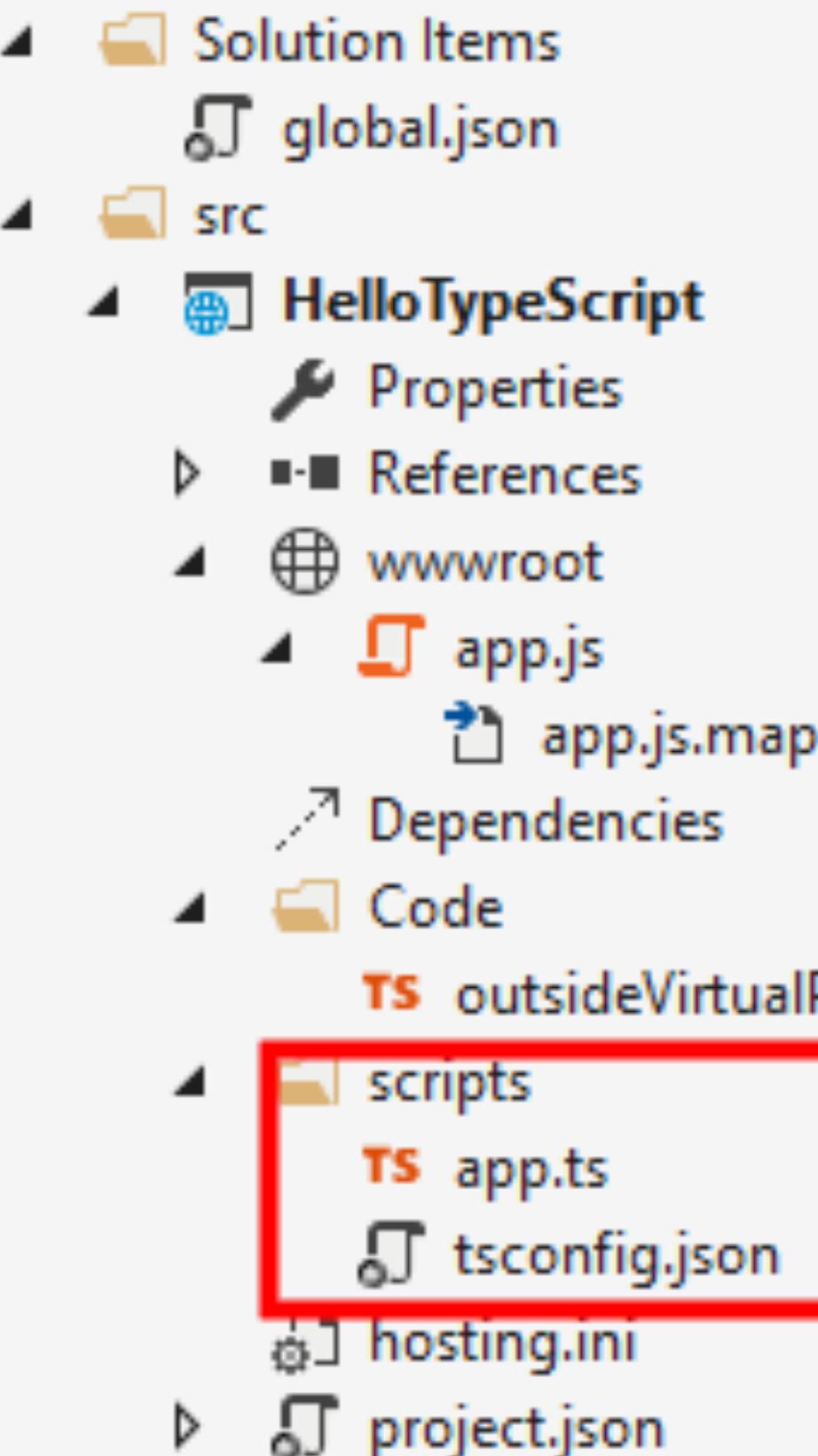
[Learn More About TypeScript](#)

[MANDATORY]

USE A CUSTOM TS.CONFIG

Search Solution Explorer (Ctrl+;)

Solution 'HelloTypeScript' (1 project)



TypeScript 2.0 is now available. [Download](#) our latest version today!

[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 `undefined`. Previously it was not possible to explicitly name these types, but `null` and `undefined` names regardless of type checking mode.

The type checker previously considered `null` and `undefined` assignable to anything. `null` and `undefined` were valid values of *every* type and it wasn't possible to specifically exclude them (which made it 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 domain of every type. They are no longer assignable to themselves and `any` (the one exception being that `undefined` is also assignable to `never`). `T` and `T | undefined` are considered synonymous in regular type checking mode, but in strict null checking mode, `T` is considered a subtype of any `T`, they are different types in strict type checking mode, and `T | undefined` permits `undefined` values. The same is true for the relationship of `T` to `T | null`.

Example



What is Prettier?

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

Why?

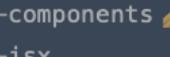
- * You press save and code is formatted
- * No need to discuss style in code review
- * Saves you time and energy
- * And more...

Works with the Tools You Use

 JavaScript
JSX
Flow
TypeScript
JSON

 HTML
Vue
Angular

 CSS
Less
SCSS

 styled-components 

 GraphQL
GraphQL Schemas

 Markdown
CommonMark
GitHub-Flavored Markdown

 MDX
YAML

 Community Plugins
Elm (via elm-format)

 Java
PHP
PostgreSQL
Ruby
Swift
TOML

⁸ [HTTPS://PRETTIER.IO](https://prettier.io)

```
1 function HelloWorld({  
2   greeting = "hello",  
3   greeted = "World",  
4   silent = false,  
5   onMouseOver  
6 }) {  
7   if (!greeting) {  
8     return null;  
9   }  
10 // TODO: Don't use random in render  
11 let num = Math.floor(Math.random() * 1e7)  
12   .toString()  
13   .replace(/\.\d+/gi, "");  
14  
15 return (  
16   <div  
17     className="HelloWorld"  
18     title={`You are visitor number ${num}`}  
19     onMouseOver={onMouseOver}  
20   >  
21     <strong>  
22       {greeting.slice(0, 1).toUpperCase() + greet  
23     </strong>  
24     {greeting.endsWith(",") ? (  
25       ""  
26     ) : (  
27       <span style={{ color: "grey" }}>, "</span>  
28     )}  
29     <em>{greeted}</em>  
30     {silent ? "." : "!"}  
31   </div>  
32 );  
33 }  
34 }  
35 }
```

TypeScript 2.0 is now available. [Download our latest version today!](#)

Quick start

Tutorials

What's New

Handbook

Basic Types

Variable Declaration

Interfaces

Classes

Functions

Generics

Enums

Type Inference

Type Compatibility

Advanced Types

Symbols

Iterators and Generators

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 { }`).

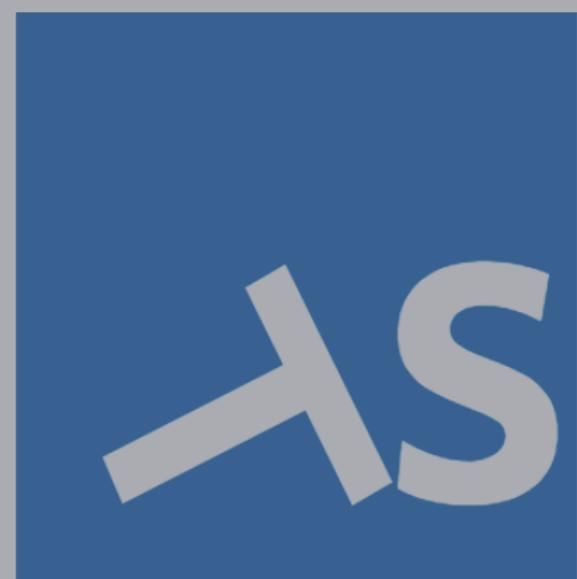
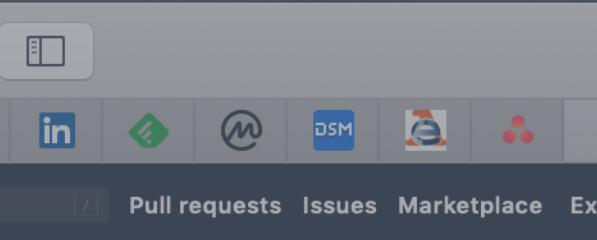
PREFER MODULES OVER NAMESPACES

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.



Typed functional programming in TypeScript

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:

[Code](#)[Issues](#)[Pull requests](#)[Marketplace](#)[Explore](#)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-specification | 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 |
| .prettierrc | Docs: fix docs |
| .prettierignore | 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 |



**THE RIGHT TOOL
FOR
THE RIGHT JOB**

A photograph of a dense forest at dusk or night. In the foreground, the dark, textured bark of a large tree trunk is visible. To the right, a vertical wooden post stands with a large, glowing green question mark painted on it. The background is filled with the silhouettes of many other trees, creating a sense of depth and mystery.

QUESTIONS?

Thank you!

Mi piace Segui Condividi ...

Gianluca Carucci
22 aprile alle ore 12:02 - 0

[COME IL LEAN THINKING PUO' FAR DIVENTARE LA TUA AZIENDA VINCENTE] [rugby]

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

SE HAI TROVATO QUESTO TALK INTERESSANTE, LEGGI ALTRI CONTENUTI SU:

Gianluca Carucci
31 dicembre 2018 · 0 · 0

E RENDERE UTILI LE RIUNIONI [clapping hands]

💡 Questa riunione è stata una perdita di tempo!! Verità, quante volte hai pronunciato qualche frase inutile nella meeting room, o chiusa l'ultima chiamata troppe forse 🤪 Siamo proprio sicuri che il problema sia la riunione?

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 un trattamento per il viso. Magari ti fa - giustamente - pensare al rugby e ti chiedi perché te ne parlando in questo contesto. 🏉 ...

Continua a leggere

