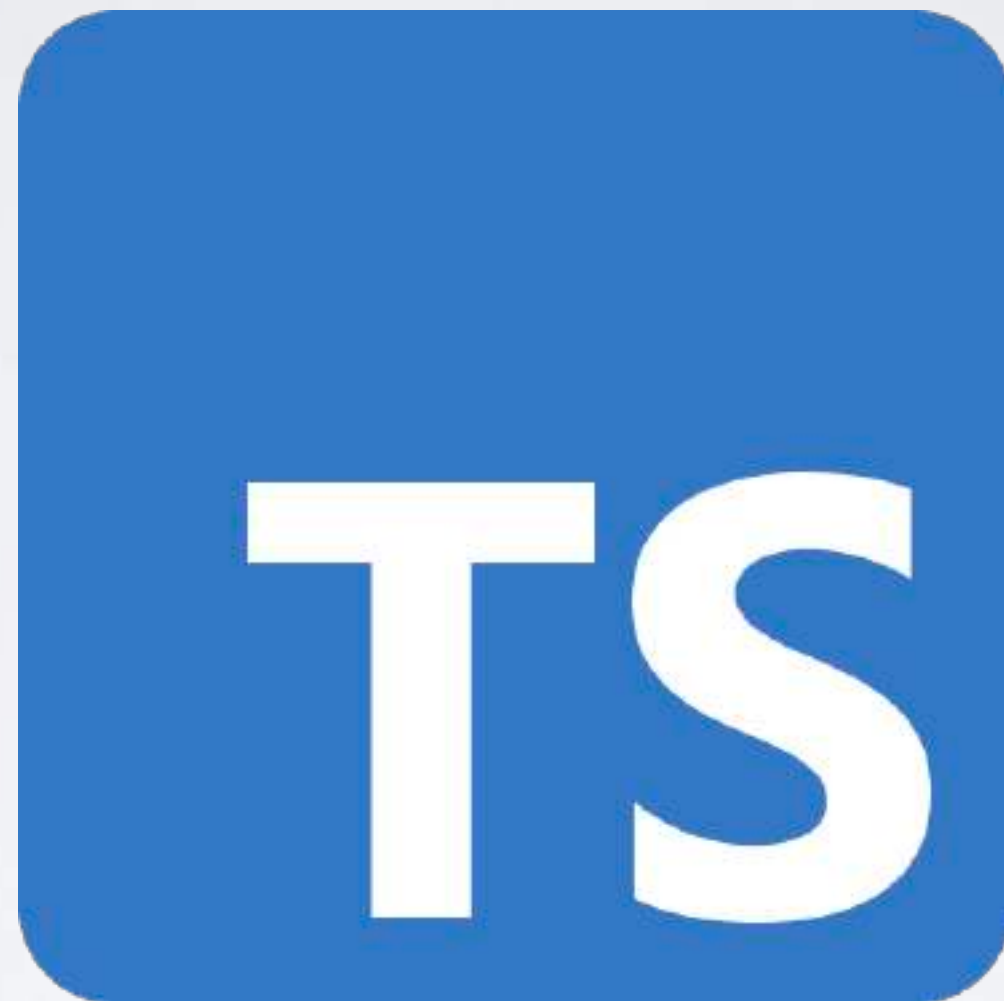


TypeScript



TypeScript

TypeScript is JavaScript with syntax for types.

TypeScript is a strongly typed programming language that builds on JavaScript, giving you better tooling at any scale

Previous Tag-Lines on the project page:

- JavaScript that scales!
- TypeScript is a language for application-scale JavaScript development.

<http://www.typescriptlang.org/>

TypeScript is very Popular

Loved by Developers

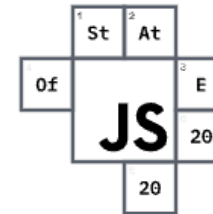


2020
Developer
Survey

Rust
TypeScript
Python



Voted **2nd most loved programming language** in the [Stack Overflow 2020 Developer survey](https://survey.stackoverflow.co/2024/technology#most-popular-technologies-language).



TypeScript was **used by 78%** of the [2020 State of JS](https://2024.stateofjs.com/en-US/usage/#js_ts_balance) respondents, with **93%** saying they would use it again.

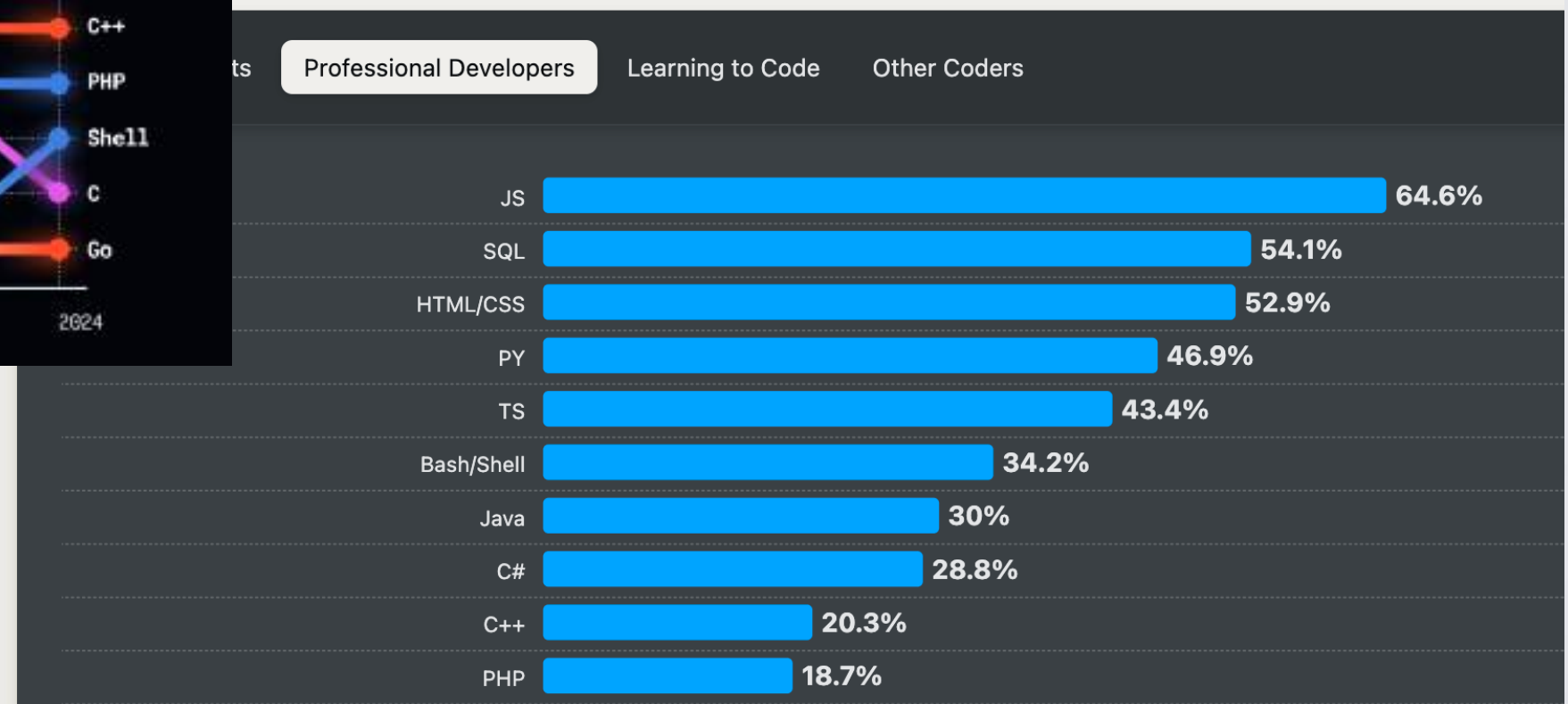
TypeScript was given the award for **"Most Adopted Technology"** based on year-on-year growth.

<https://survey.stackoverflow.co/2024/technology#most-popular-technologies-language>

https://2024.stateofjs.com/en-US/usage/#js_ts_balance

<https://innovationgraph.github.com/global-metrics/programming-languages>

The Rise of TypeScript



<https://github.blog/news-insights/octoverse/octoverse-2024/#the-most-popular-programming-languages>

<https://survey.stackoverflow.co/2024/technology#most-popular-technologies-language-prof>

The History of TypeScript



The Story of TypeScript (6:26)
<https://www.youtube.com/watch?v=EUIM3wx546o>



TypeScript Origins: The Documentary (1:21:35)
<https://www.youtube.com/watch?v=U6s2pdxebSo>

TypeScript and the dawn of gradual types:
<https://github.com/readme/featured/typescript-gradual-types>

There are Alternatives ...



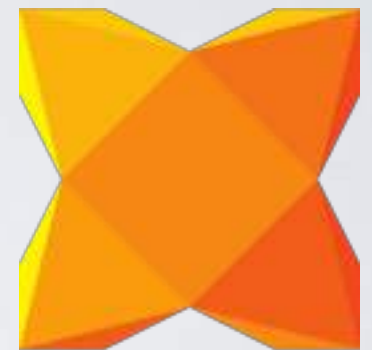
<https://rescript-lang.org/>



<https://flow.org/>



<https://www.purescript.org/>



HAXE

<https://haxe.org/>

... but none of them has the momentum of TypeScript.

Developer platforms get better at a rate proportional to the number of developers using them.

Geoff Schmidt, JavaScript State of the Union 2015

<https://www.youtube.com/watch?v=8G2SMVIUNNk>

ECMAScript Versions

- JavaScript: Initial Release in Netscape 2, 1995
- ECMAScript 1: 1997
- ECMAScript 2: 1998
- ECMAScript 3: 1999
- ~~ECMAScript 4: abandoned~~
- ECMAScript 5: 2009
- ECMAScript 2015: 2015
- ECMAScript 2016
- ECMAScript 2017
- ...
- ECMAScript 2024
- ECMAScript 2025

 2012: TypeScript

JavaScript was never engineered for large applications. It was intended for 100–1000 lines of codes.

Today we build apps with millions of lines of code. Very large JavaScript code bases tend to become “read-only”.

– Anders Hejlsberg, Build 2016

Why TypeScript?



dynamically typed



statically typed



"gradual" typed
(best of both worlds?)

TypeScript and the dawn of gradual types:
<https://github.com/readme/featured/typescript-gradual-types>

Gradual Typing: https://en.wikipedia.org/wiki/Gradual_typing

ECMAScript Proposal (2022): type annotations
<https://github.com/tc39/proposal-type-annotations>



David K. 
@DavidKPiano



I love TypeScript. It keeps me from coding too fast.

4:32 PM · Feb 19, 2021 · Twitter Web App

118 Retweets **23** Quote Tweets **1,615** Likes



David K.  @DavidKPiano · Feb 19



Replying to [@DavidKPiano](#)

JavaScript = move fast and break things

TypeScript = move slow and prevent things from breaking



15



33




280



<https://twitter.com/DavidKPiano/status/1362787204058263558>



Kent C. Dodds 

@kentcdodds



“TypeScript isn’t making your life worse. It’s just showing you how bad your life already is”

— Me, in a workshop right now explaining how annoying form elements are to get actual runtime safety.

TypeScript is "abstract"!

You can't "fix" a bug with TypeScript!

```
interface Person {
  firstName: string;
  movies: string[];
}

async function fetchLuke(): Promise<Person> {
  const lukeResponse = await fetch(`https://swapi.info/api/people/1`);
  const luke = await lukeResponse.json() as Person;
  return luke
}

async function main() {
  console.log("Starting ...");
  const luke = await fetchLuke();

  // 💥 KABOOM! - the following line throws a TypeError at runtime
  console.log(`Luke appears in ${luke.movies.length} movies.`);
}

main();
```

But correctly used TypeScript does prevent bugs ...



west, donavon west
@donavon



If you lie to TypeScript, TypeScript will lie to you.

7:23 PM · Aug 24, 2021 · Twitter Web App

<https://twitter.com/donavon/status/1430219301500358658>

using axios with TypeScript

```
const data = await axios.get<Person>(BACKEND_URL);
```

using ky with TypeScript

```
const data = await ky.get(BACKEND_URL).json<Person>();
```

```
interface Person {  
  firstName: string;  
  lastName: string;  
}
```

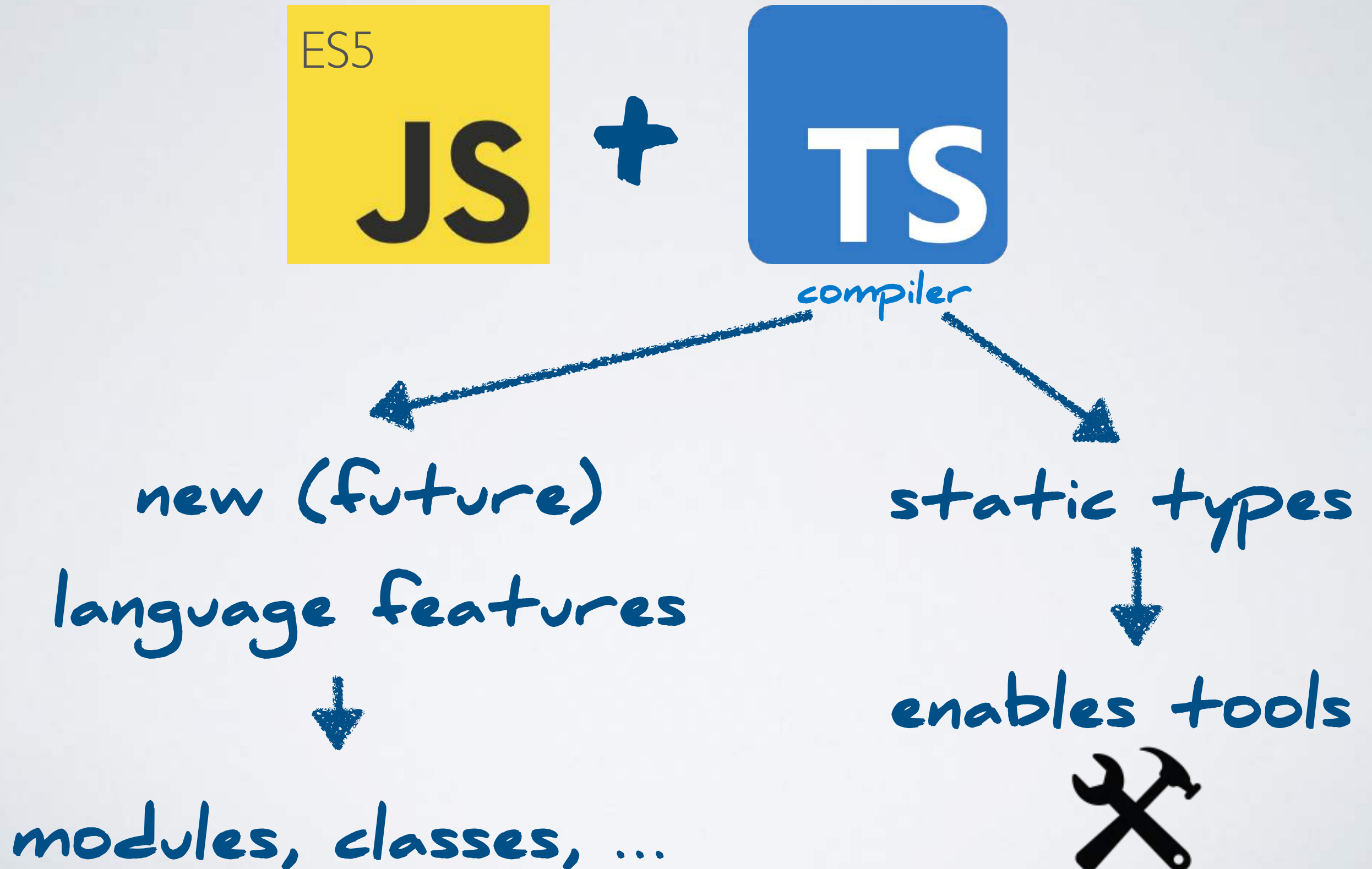
using Angular HttpClient

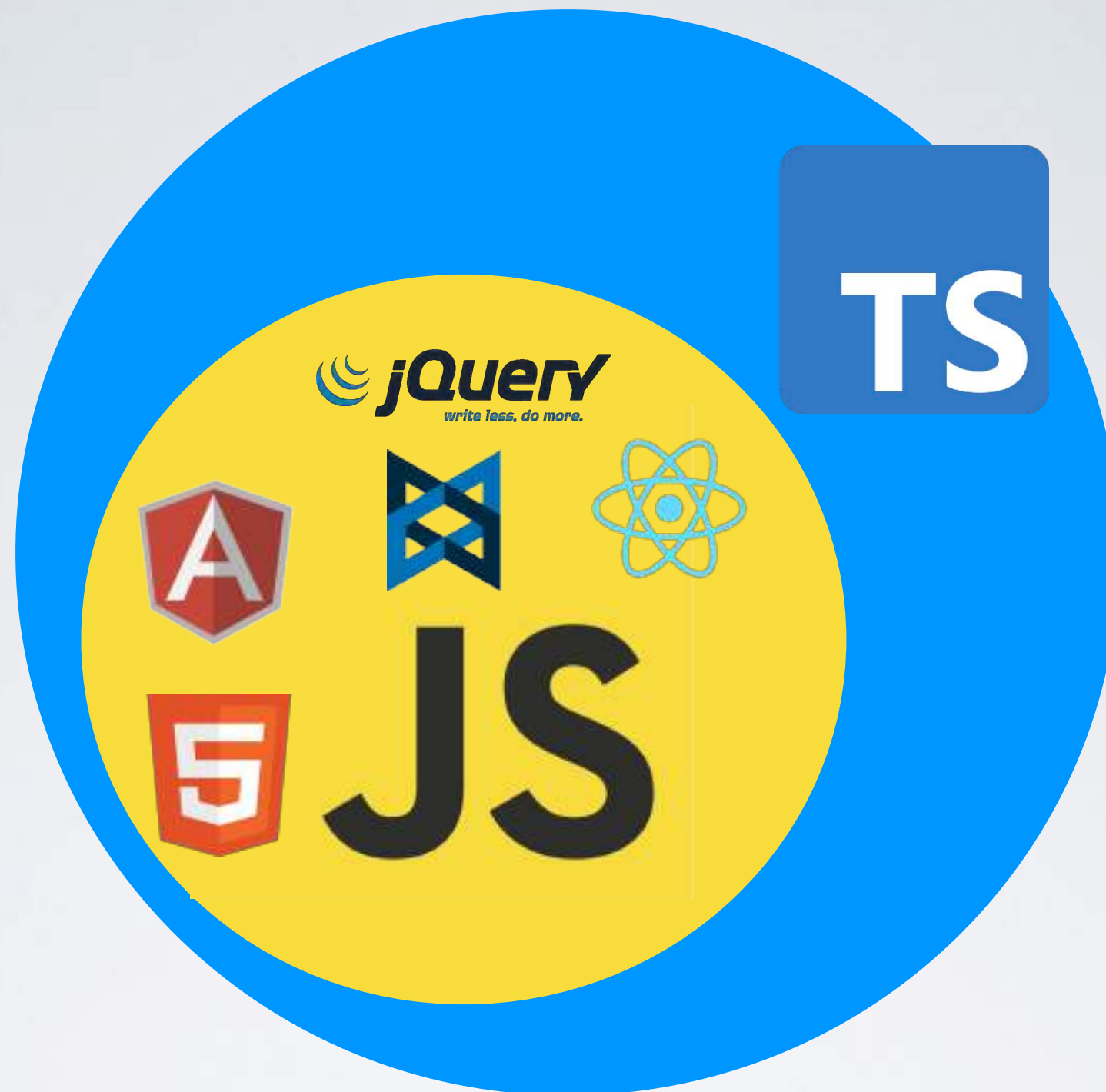
```
fetchData(): Observable<Person> {  
  return this.http.get<Person>(BACKEND_URL);  
}
```

The type information is only for the TypeScript compiler at build time. At runtime there is no guarantee or check that the network call really returns objects of the specified type.

You should only use **interface** or **type** as *type arguments* for backend access (not **class**).

The Goal of TypeScript in 2012





"TypeScript is a superset of JavaScript."

Any valid JavaScript (theoretically) is valid TypeScript. But TypeScript adds syntax for types, which is not valid JavaScript.

"TypeScript is a subset of JavaScript."

TypeScript throws build-time errors for valid JavaScript, constraining the "flexibility" of JavaScript.

TS is a superset of JS. Start with your existing JS and "enhance" it ...



It starts with JavaScript.

It ends with JavaScript.



After transpilation it's pure JS again. At runtime there is nothing left of TS!

TypeScript "Compilation"

At runtime TypeScript is just JavaScript.

Type information is stripped before runtime, typically at build-time.

TypeScript becomes JavaScript via the delete key.

```
type Result = "pass" | "fail"

function verify(result: Result) {
  if (result === "pass") {
    console.log("Passed")
  } else {
    console.log("Failed")
  }
}
```

TypeScript file.

```
type Result = "pass" | "fail"

function verify(result: Result) {
  if (result === "pass") {
    console.log("Passed")
  } else {
    console.log("Failed")
  }
}
```

Types are removed.

```
function verify(result) {
  if (result === "pass") {
    console.log("Passed")
  } else {
    console.log("Failed")
  }
}
```

JavaScript file.

TypeScript Today

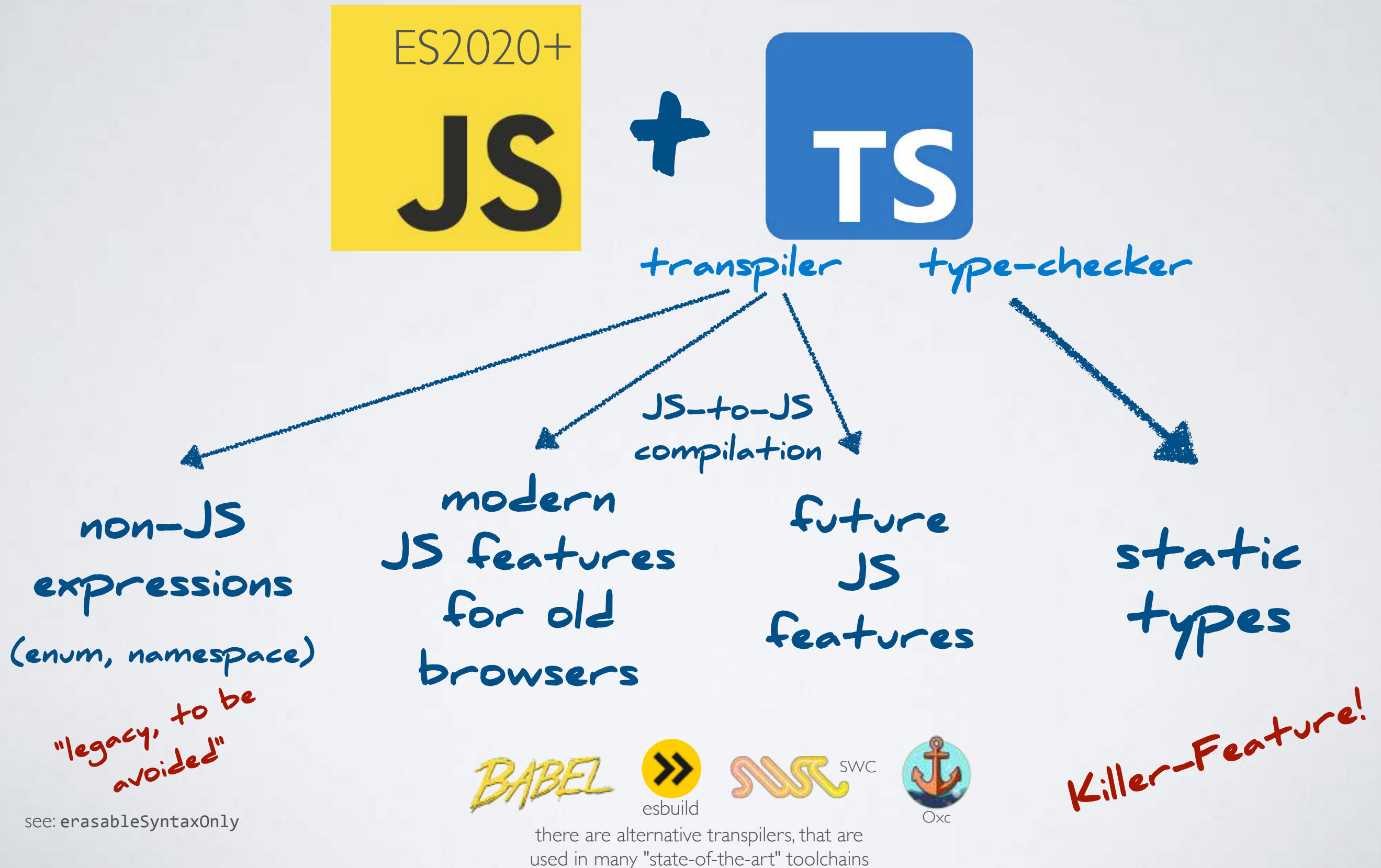
- ... is a compiler that provides modern (ES2015+) language features for older Browsers
- ... is a compiler that provides some future ESnext language features (i.e. decorators, **using** declarations ...)
- ... is a compiler that adds an optional static type system on top of JavaScript.
- ... is also compiler that provides a few constructs that are not available in JavaScript (enums, namespaces) - however today it is "good practice" not to use these.
Since TypeScript 5.8 this can be switched of with `erasableSyntaxOnly: true`

} Transpiler

} Type Checker

"legacy"

TypeScript Today



TypeScript is currently rewritten in Go

(announced March 2025)

<https://devblogs.microsoft.com/typescript/typescript-native-port/>

Expectations:

- The TypeScript compilation step in the build pipeline becomes 10x faster
- The "IDE experience" will become 10x faster
- Your code does not have to change

Try it: <https://github.com/microsoft/typescript-go>

TypeScript Design Goals

<https://github.com/Microsoft/TypeScript/wiki/TypeScript-Design-Goals>

Impose no runtime overhead on emitted programs.

Align with current and future ECMAScript proposals.

Preserve runtime behavior of all JavaScript code.

Avoid adding expression-level syntax.

Use a consistent, fully erasable, structural type system.

Consequences:

- integrating existing JavaScript is easy.
- migrating from JavaScript is easy.
- *You can't be proficient in TypeScript without (deep) knowlege of JavaScript!*



Myth:
There can be only one ...

JavaScript VS TypeScript: Which is better? (2020 Updated)



Infinijith Apps & Technologies Mar 10, 2020 · 6 min read



<https://shrtm.nu/RmGK>

Is JavaScript Becoming TypeScript?

What might the future hold in store?



Mahdhi Rezvi [Follow](#)

Feb 4 · 9 min read



<https://shrtm.nu/Q0Cx>

A Post-Mortem in 5 Acts, of How Microsoft Privatized Open Source, killing JavaScript in the Process

After Microsoft's blitzkrieg take-over, the Open Source JavaScript community, as we know it, is coming to an end.



Alex Kleydints Feb 6 · 9 min read ★



<https://shrtm.nu/KVWLb>

It is TypeScript *with* JavaScript ...

... not TypeScript *versus* JavaScript!
... but TypeScript *on top* of JavaScript



A bet on TypeScript is a bet on JavaScript!

TypeScript and JavaScript are allies in trying to make cool things in JavaScript.

The TypeScript team doesn't want to be in-front of JavaScript development
but to let JS lead and TS follow.

- Orta Therox from the TypeScript team
<https://www.youtube.com/watch?v=8qm49TyMUPI>

We don't see it as our job to add new features in the core language. We closely track ECMA Script. We innovate in the type system, which is a "development-time-only-thing".

- Anders Hejlsberg 2018
<https://www.youtube.com/watch?v=ET4kT88JRXs>



Myth:
TypeScript makes my program
typesafe like C++, C# or Java.

TypeScript is purely a build-time construct.

The TypeScript compiler ensures type safety for all the code it "sees" at compile time.

If your program interacts with code the TypeScript compiler can't see, like

- network
- local storage
- 3rd party libraries

... then you have the untyped behavior of JavaScript.

The type system of TypeScript is optional. Programmers can always "opt out" and work with untyped JavaScript.



Myth : Angular is the best choice if I want to use TypeScript.

TypeScript sees itself as the Switzerland for JavaScript Frameworks

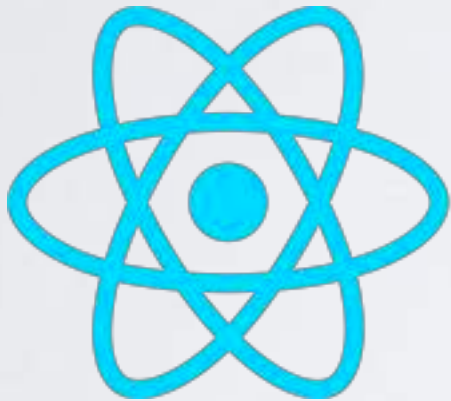
- Anders Hejlsberg, TypeScript: Static types for JavaScript

<https://www.youtube.com/watch?v=ET4kT88JRXs>



Angular is written in TypeScript and the whole Angular ecosystem and its community has fully embraced TypeScript.

But Angular versions have a tighter coupling to TypeScript versions than other frameworks:
<https://angular.dev/reference/versions#actively-supported-versions>



TypeScript support is especially strong for React, since React does not have its own templating "language". JSX is directly supported by TypeScript in a typesafe way.



Vue 3 was a full rewrite in TypeScript. One of the goals was to provide better support for TypeScript projects.



Myth:
TypeScript is easy

The Challenge of TypeScript

The start is easy:

```
1 export type Todo = {
2   id: number;
3   title: string;
4   completed: boolean;
5 }
6
7 const todoItem: Todo = {
8   id: 1234, title: 'Buy milk', completed: false
9 }
10
11 const todoItem2 = {
12   id: 5, title: 'Go to gym', completed: false
13 }
14
15 const todos: Todo[] = [todoItem, todoItem2];
```

... but it can become quite complex:

```
export type OmitK<T, K> = Pick<T, Exclude<keyof T, K>>;

export interface BiBaseEvent {
  msgId: string;
  src: number;
  hosting: string;
}

export type BiBaseParams<T extends BiBaseEvent> = Omit<T, 'src' | 'hosting'> & { viewMode: string };

export interface BaseComponentProps {
  id: string;
  disabled: boolean;
  className?: string;
  onClick?: (id: string, event: React.MouseEvent<HTMLButtonElement>) => void;
}
```

<https://medium.com/swlh/understanding-the-weird-parts-of-typescript-20c0fe26d314>

```
@Component({
  moduleId: module.id,
  selector: 'signup',
  templateUrl: 'signup.component.html',
  directives: [REACTIVE_FORM_DIRECTIVES],
  providers: [AuthenticationService]
})
export class SignupComponent implements
  [ts]
Argument of type '(((control: AbstractControl) => { [key: string]: boolean; }) | ((control:
AbstractControl) => { [key: string]: boolean; }) | ((control: AbstractControl) => {
Type '(((control: AbstractControl) => { [key: string]: boolean; }) | ((control: AbstractControl) => {
[k...]' is not assignable to type 'ValidatorFn'.
Type '(control: AbstractControl) => { [key: string]: boolean; }' is not assignable to type
'ValidatorFn'.
Types of parameters 'control' and 'c' are incompatible.
Type 'AbstractControl' is not assignable to type 'AbstractControl'.
Types of property 'validator' are incompatible.
Type 'ValidatorFn' is not assignable to type 'ValidatorFn'.
Types of parameters 'c' and 'c' are incompatible.
Type 'AbstractControl' is not assignable to type 'AbstractControl'.
Types of property 'validator' are incompatible.
Type 'ValidatorFn' is not assignable to type 'ValidatorFn'.
Types of parameters 'c' and 'c' are incompatible.
Type 'AbstractControl' is not assignable to type 'AbstractControl'.
(method) ValidationService.emailValidator(control: AbstractControl): {
[key: string]: boolean;
}

signupForm: FormGroup;
submitted = false;
errorMessage: string;
accountTypes: SelectItem[];
selectedType: any;

constructor(public router: Router, private
this.accountTypes = [];
this.accountTypes.push({label: 'Indi
this.accountTypes.push({label: 'Orga
this.signupForm = this.fb.group({
  'email': ['', Validators.compose([Validators.required, ValidationService.emailValidator])],
  'username': ['', Validators.compose([Validators.required, Validators.minLength(6)])],
```

```
[ts]
Argument of type '{ declarations: (typeof ConversationPage)[];
imports: (ModuleWithProviders | typeof moment)[]; en...' is no
t assignable to parameter of type 'NgModule'.
Types of property 'imports' are incompatible.
Type '(ModuleWithProviders | typeof moment)[]' is not assig
nable to type '(any[] | Type<any> | ModuleWithProviders)[]'.
Type 'ModuleWithProviders | typeof moment' is not assigna
ble to type 'any[] | Type<any> | ModuleWithProviders'.
Type 'typeof moment' is not assignable to type 'any[] |
Type<any> | ModuleWithProviders'.
Type 'typeof moment' is not assignable to type 'Modul
eWithProviders'.
```

<https://stackoverflow.com/questions/51081676/not-able-to-use-moment-mini-ts-lib>

<https://stackoverflow.com/questions/38954021/angular-2-error-type-abstractcontrol-is-not-assignable-to-type-abstractcont>

Adam Rackis

@AdamRackis

...

* wife flies to DC for the week *

Expectation: "Tons of free time. Gonna catch up on some reading. Gonna catch up on some TV 🚀"

Reality:

```
type TuplesOverlapPerfectly<T, U> = T extends []
  ? true
  : U extends []
  ? true
  : T extends [infer THead, ...infer TRest]
  ? U extends [infer UHead, ...infer URest]
    ? UHead extends THead
      ? THead extends UHead
        ? TuplesOverlapPerfectly<TRest, URest>
          : false
        : false
      : false
    : false;

type TupleOneIsLongerOrEqual<T, U> = U extends []
  ? true
  : T extends []
  ? false
  : T extends [infer _, ...infer TRest]
  ? U extends [infer _, ...infer URest]
    ? TupleOneIsLongerOrEqual<TRest, URest>
      : never
    : never;

type GetBroaderTuple<T, U> = TuplesOverlapPerfectly<T, U> extends true
  ? TupleOneIsLongerOrEqual<T, U> extends true
    ? T
    : U
  : never;
```

6:31 AM · May 8, 2024 · 1M Views

DHH   @dhh · May 9

...

Never seen a better ad for ditching TypeScript. You don't have to live like this!!

https://twitter.com/i/bookmarks?post_id=1788597500192133257

TypeScript

Project Setup

Global installation: ~~avoid!~~
`npm install -g typescript`

Local installation:
`npm init -y`
`npm install -D typescript`

Configuration:
`npx tsc --init`
`edit tsconfig.json`

Usage:
run via: `npx tsc`
or via npm script: `tsc`

(modern frontend templates are setting up TypeScript, so typically you don't have to do this manually today)

tsconfig setup

The TypeScript compiler is configured in `tsconfig.json`

```
{
  "compilerOptions": {
    "strict": true,
    "noImplicitOverride": true,
    "noPropertyAccessFromIndexSignature": true,
    "noImplicitReturns": true,
    "noFallthroughCasesInSwitch": true,
    "skipLibCheck": true,
    "isolatedModules": true,
    "experimentalDecorators": true,
    "importHelpers": true,
    "target": "ES2022",
    "module": "preserve"
  }
}
```

(example generated by Angular CLI v20)

Special mention: **erasableSyntaxOnly**

<https://www.typescriptlang.org/docs/handbook/release-notes/typescript-5-8.html#the---erasablesyntaxonly-option>

<https://www.typescriptlang.org/tsconfig>

<https://www.typescriptlang.org/docs/handbook/tsconfig-json.html>

Type System Features

- Zero Cost: Static types only during development time, *no effect at runtime*
- Type inference
- Gradual typing
- Configurable type checking

Using Basic Types

Types are an optional, opt-in feature in TypeScript.

Basic Types:

```
let b:boolean = true;
let n:number = 42;
let s:string = 'Hello!';
let a1: number[] = [1,2,3];
let a2:Array<string> = ['Hello','World', '!'];
enum ArrowDirection { Up, Down, Left, Right}
let e: ArrowDirection = ArrowDirection.Up;
```

Basic type checking:

```
let x:string = 'Hello World';
x = 42; // => compiler error
```

Type inference:

```
let x = 'Hello World'; // the type of x is 'string'
x = 42; // => compiler error
```

The compiler will infer types if possible. It's considered a 'good style' to omit type annotations if they can be derived.

Built-In Types

TypeScript has built-in support for JavaScript types:

```
[1, 2, 3, 4].reduce(  
  (acc : {sum: number} , e : number ) => {  
    return {sum: acc.sum + e}  
  }, {sum: 0});
```

TypeScript has built-in support for browser APIs:

```
document.addEventListener( type: 'keydown', listener: function(event : KeyboardEvent) {  
  console.log(event.clientX, event.clientY);  
});
```

```
document.querySelector<HTMLDivElement>('#app')!.innerHTML = '';
```

TS2551: Property 'innerHTML' does not exist on type 'HTMLDivElement'. Did you mean 'innerHTML'?

lib.dom.d.ts(9130, 5): 'innerHTML' is declared here.

Rename reference ↗ ↕ ↶ More actions... ↗ ↕ ↶

Installing 3rd-Party Type Definitions

Some libraries are not written in TypeScript.
Their type definitions can still be available via npm:

```
npm install @types/jquery
```

(**@types** is called a “scope” in npm)

The type definitions are maintained in the Definitely Typed repository:

- <https://github.com/DefinitelyTyped/DefinitelyTyped/tree/master/types>
- <https://definitelytyped.org/>

Advanced Types

Union Types:

```
let pet: IBird | IFish = {fly() {}, layEggs() {}};  
pet = new Fish();
```

Intersection Types:

```
let flyingFish: IFish & IBird = {swim() {}, fly() {}, layEggs() {}};  
flyingFish.swim();
```

Type Guards:

```
(pet as IFish).swim();  
(<IFish>pet).swim();  
  
if (pet instanceof Fish){  
    pet.swim();  
}
```

User defined type-guards:

```
function isFish(pet: Fish | Bird): pet is Fish {  
    return (pet as any).swim !== undefined;  
}
```

```
if (isFish(pet)) {  
    pet.swim();  
} else {  
    pet.fly();  
}
```

any Type

Types are optional. TypeScript allows you to gradually opt-in and opt-out of type-checking during compilation.

```
let v: any = 42;  
v = 'Hello!'; // anything can be assigned to v  
  
let n: number = 43;  
n = v; // v can be assigned to anything!!!
```

Variables have the implicit type **any** if TypeScript does not know its type.

Typing can be enforced by setting
noImplicitAny: true in **tsconfig.json**

never & unknown Types

Some functions *never* return a value:

```
function fail(msg: string): never {  
    throw new Error(msg);  
}
```

The *unknown* type represents any value. But it is not legal to do anything with an *unknown* value:

```
function f1(a: any) {  
    a.b(); // OK  
}  
function f2(a: unknown) {  
    a.b(); //ERROR: Object is of type 'unknown'.  
}
```

Type Narrowing

```
function padLeft(padding: number | string, input: string): string {  
  if (typeof padding === "number") {  
    return " ".repeat(padding) + input; // -> padding is of type "number"  
  }  
  return padding + input; // -> padding is of type "string"  
}
```

There are many mechanisms how TypeScript performs type narrowing:

<https://www.typescriptlang.org/docs/handbook/2/narrowing.html>

```
function isFish(pet: Fish | Bird): pet is Fish {  
  return (pet as Fish).swim !== undefined;  
}
```

<https://www.typescriptlang.org/docs/handbook/2/narrowing.html#using-type-predicates>

Function Types

A function has a type defined by its signature

```
let operation: string = 'add';

let mathFunc: (x: number, y:number) => number;

function add(x: number, y:number) : number {
    return x + y;
}

const subtract = (x: number, y:number):number => x - y;

if (operation === 'add')
    mathFunc = add;
else
    mathFunc = subtract;

console.log(mathFunc(4,2))
```

See also: <https://kentcdodds.com/blog/typescript-function-syntaxes>

Note: functions with fewer parameters are assignable to functions that take more parameters.

<https://github.com/Microsoft/TypeScript/wiki/FAQ#why-are-functions-with-fewer-parameters-assignable-to-functions-that-take-more-parameters>

Function Overloads

In my experience a feature that is rarely used / useful.

```
// declaration
function doIt(): number;
function doIt(arg1: number): number;
function doIt(arg1: number, arg2: string): number;
function doIt(arg1?: number, arg2?: string): number {
    return 42;
}

// usage options
doIt();
doIt(44);
doIt(44, 'test');
```

overload signatures

implementation signature

The implementation signature must be compatible to all overload signatures.

Classes & Interfaces

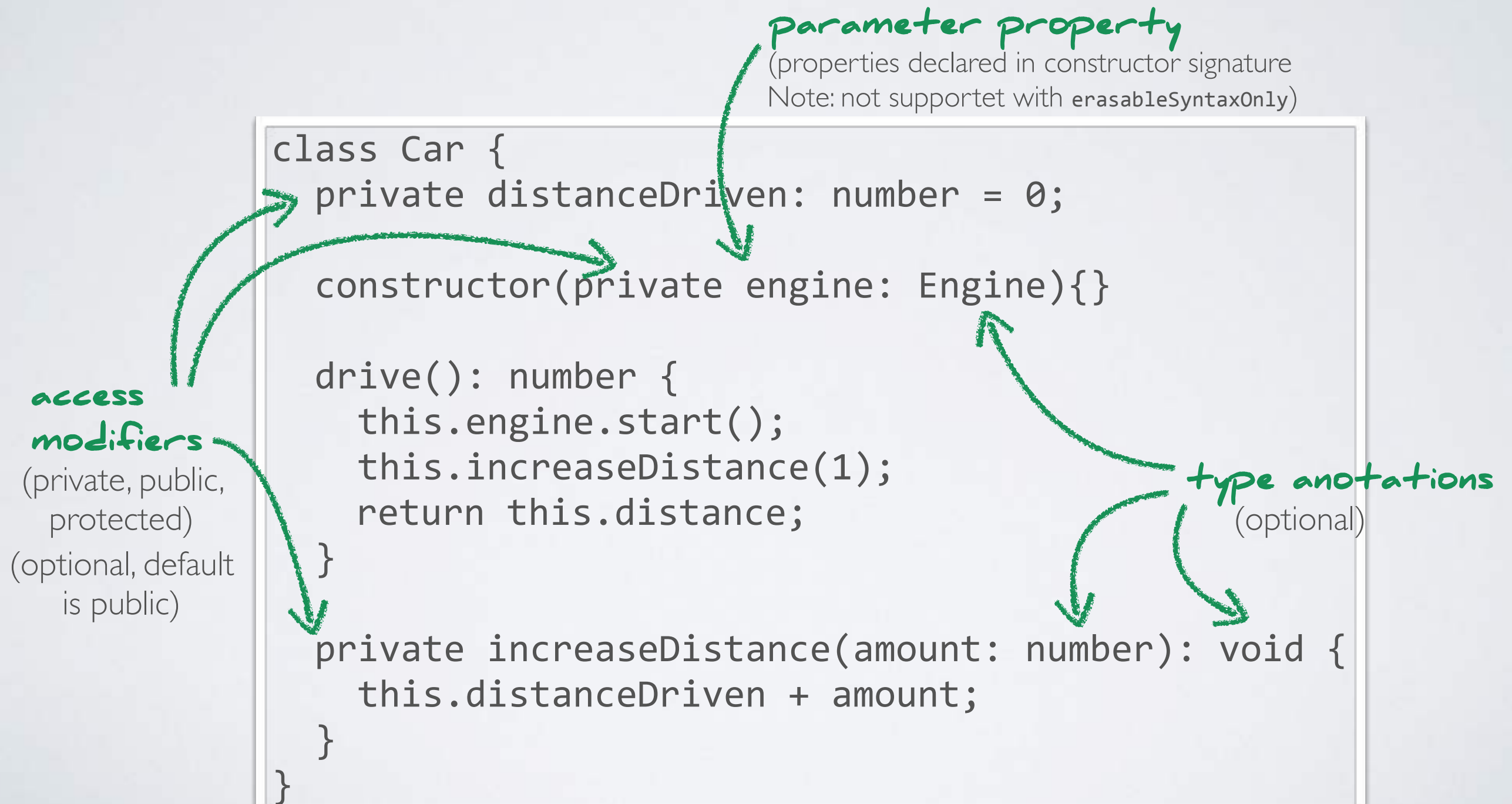
With classes & interfaces we can create custom types in TypeScript

```
interface IPerson {  
    name: string,  
}  
  
class Person implements IPerson {  
    name: string = 'Tyler';  
    private age: number = 42;  
  
    getInfo() {  
        console.log(this.name); // inspect `this`  
        return this.name + this.age;  
    }  
}  
  
const p1: IPerson = new Person();  
const p2: Person = new Person();
```

Interfaces are a pure build-time construct. They are not present at runtime!

TypeScript Classes

Differences to JavaScript classes:



The Structural Type System

The structure of an objects defines the compatibility with a type, the object does not have to be explicitly declared as a type (aka. "Duck Typing")

```
interface IPerson {  
    name: string,  
    age: number  
}  
  
const p3: IPerson = {  
    name: 'Tyler',  
    age: 42  
};
```

Type Aliases

<https://www.typescriptlang.org/docs/handbook/advanced-types.html#type-aliases>

Type Aliases give a name to a type.

Describe the shape of an object:

```
type Person = {  
  firstName: string;  
  lastName: string;  
  greet: (message: string) => string  
}
```

Name a union type:

```
type ID = number | string;
```

Enums in TypeScript

In many cases Enums are not recommended!
Use union types instead!

```
enum Status {  
  Admin,  
  User,  
  Moderator,  
}
```

```
type Status = "Admin" | "User" | "Moderator";
```

prefer!

```
enum Status {  
  Admin = "Admin",  
  User = "User",  
  Moderator = "Moderator",  
}
```

avoid!

emitted
JavaScript

```
var Status;  
(function (Status) {  
  Status["Admin"] = "Admin";  
  Status["User"] = "User";  
  Status["Moderator"] = "Moderator";  
})(Status || (Status = {}));
```

Note:

enums can't be used native

Node / Bun / Deno and with

`erasableSyntaxOnly: true`

Reason: <https://fettblog.eu/tidy-typescript-avoid-enums/>
More Info: <https://stackoverflow.com/a/60041791/32749>

In modern TypeScript, you may not need an enum when an object with **as const** could suffice:

<https://www.typescriptlang.org/docs/handbook/enums.html#objects-vs-enums>

```
const ODirection = {  
  Up: 0,  
  Down: 1,  
  Left: 2,  
  Right: 3,  
} as const;  
type Direction = typeof ODirection[keyof typeof ODirection];
```

```
function run(dir: Direction) {}  
  
run(ODirection.Right);
```


Types Aliases vs. Interfaces

In most cases type alias is interchangeable with an interface.

equivalent

```
type Person = {  
  firstName: string;  
  lastName: string;  
  greet: (message: string) => string  
}
```

```
interface Person {  
  firstName: string;  
  lastName: string;  
  greet: (message: string) => string  
}
```

Special cases, not possible with interfaces:

```
type Second = number;
```

alias for primitive type

```
type Status = "Admin" | "User" | "Moderator";
```

union types

A simple strategy to decide what to use:
use interfaces to describe objects, use types for everything else.

Main difference: Interfaces can be defined/extended in multiple files. Types can't.

Differences: <https://www.typescriptlang.org/docs/handbook/2/everyday-types.html#differences-between-type-aliases-and-interfaces>

TypeScript recommends to use interfaces over type aliases:

Better error messages: <https://www.typescriptlang.org/play?q=378#example/types-vs-interfaces>

Performance: <https://www.typescriptlang.org/play?q=378#example/types-vs-interfaces>

Marking values as readonly with `as const`

```
const obj = {  
  direction: "up" as const,  
  foo: {  
    bar: 42,  
  },  
} as const;  
obj.foo.bar = 43;  
console.log(obj.direction);
```

compile error

*type is "up"
not string!*

```
const arr = [ 42, "test" ] as const;  
arr.push(43);  
  
function fn(num: number){  
  console.log(num);  
}  
  
fn(arr[0]);  
fn(arr[1]);
```

compile error

ok, because type is number

compile error because type is "test"/string

<https://www.totaltypescript.com/concepts/as-const>

<https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes-func.html#readonly-and-const>

Creating Types from Types

"Metaprogramming with Types"

<https://www.typescriptlang.org/docs/handbook/2/types-from-types.html>

Example:

```
const person = {
  firstName: 'Jonas',
  age: 42,
  address: {street: 42}
};

type T1 = typeof person;
type T2 = keyof typeof person;
type T4 = T1['address']['street'];

function getVal(){
  return new Promise<T1>(resolve => {
    resolve(person)
  });
}

type T3 = keyof Awaited<ReturnType<typeof getVal>>;
```

<https://www.zhenghao.io/posts/type-programming>

<https://medium.com/@imkelen009/advanced-usage-of-typescript-%E4%B8%80-5b102a459b8d>

<https://obaranovskyi.medium.com/10-typescript-features-you-might-not-be-using-yet-or-didnt-understand-d1f2888ea45>

Crazy: TypeScript Types can run DOOM!

(game from 1993)

<https://www.youtube.com/watch?v=0mCsluv5FXA>

To prove that TypeScript Types are Turing Complete, someone implemented a TypeScript program that is running DOOM:

calling **tsc** returns a TypeScript object that represents the ASCII-Art of a Frame of DOOM.

Rendering the first frame of DOOM took running the program for 12 days with 20 million type instantiations per second ...

<https://github.com/MichiganTypeScript/typescript-types-only-wasm-runtime>

TypeScript Decorators

Decorators are an upcoming ECMAScript feature that allow us to customize classes and their members in a reusable way.

Decorators are heavily used in Angular.

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

  @loggedMethod
  greet() {
    console.log(`Hello, my name is ${this.name}.`);
  }
}

const p = new Person("Ron");
p.greet();
```

```
function loggedMethod(originalMethod: any,
                      context: ClassMethodDecoratorContext) {
  const methodName = String(context.name);

  function replacementMethod(this: any, ...args: any[]) {
    console.log(`LOG: Entering method '${methodName}'.`)
    const result = originalMethod.call(this, ...args);
    console.log(`LOG: Exiting method '${methodName}'.`)
    return result;
  }

  return replacementMethod;
}
```

```
// Output:
//
//   LOG: Entering method.
//   Hello, my name is Ron.
//   LOG: Exiting method.
```

A wide-angle photograph of the Golden Gate Bridge in San Francisco, taken from a low angle looking up at the tower. The bridge's orange-red steel structure is prominent against a blue sky with light clouds. The water of the bay is visible below, and the city skyline is in the distance.

JS

Java

TS

Runtime Type-Safety

(typesafe backend access and input validation)