


TYPESCRIPT: COLLECTED WORKS

[**https://github.io/przemyslawjanpietrzak**](https://github.io/przemyslawjanpietrzak)

[**https://twitter.com/przemyslawjanp**](https://twitter.com/przemyslawjanp)



😊 Set status

Przemyslaw Jan Beigert

przemyslawjanpietrzak

★ PRO

Edit profile








Software engineer, enthusiast of new technologies (but only if are better than old ones). Open source and functional programming fan.

📍 Poznan

✉ przemyslaw.jan.beigert@protonmail...

🌐 <https://twitter.com/przemyslawjanp>

Organizations



OverviewRepositories 47Projects 0Stars 239Followers 16Following 36

Pinned

Customize your pins

rembrandt

Simple functional UI framework written in Reasonml.

OCaml★ 50

pyMonet

High abstract python library for functional programming. Contains algebraic data structures known (or unknown) from Haskell or Scala.

Python★ 21

RxTowerDefense

Tower defense engine written in TypeScript with rx.js6, three.js, and pattern from Cycle.js.

TypeScript★ 7🔗 1

stanza.io-examples-tests

Examples of communication with stanza.io library by XMPP protocol, as jasmine unit tests

JavaScript★ 2🔗 1

dotfiles

Script for prepare fresh ubuntu instance to developers needs, like python, node, docker, vscode etc etc, etc.

Emacs Lisp★ 2

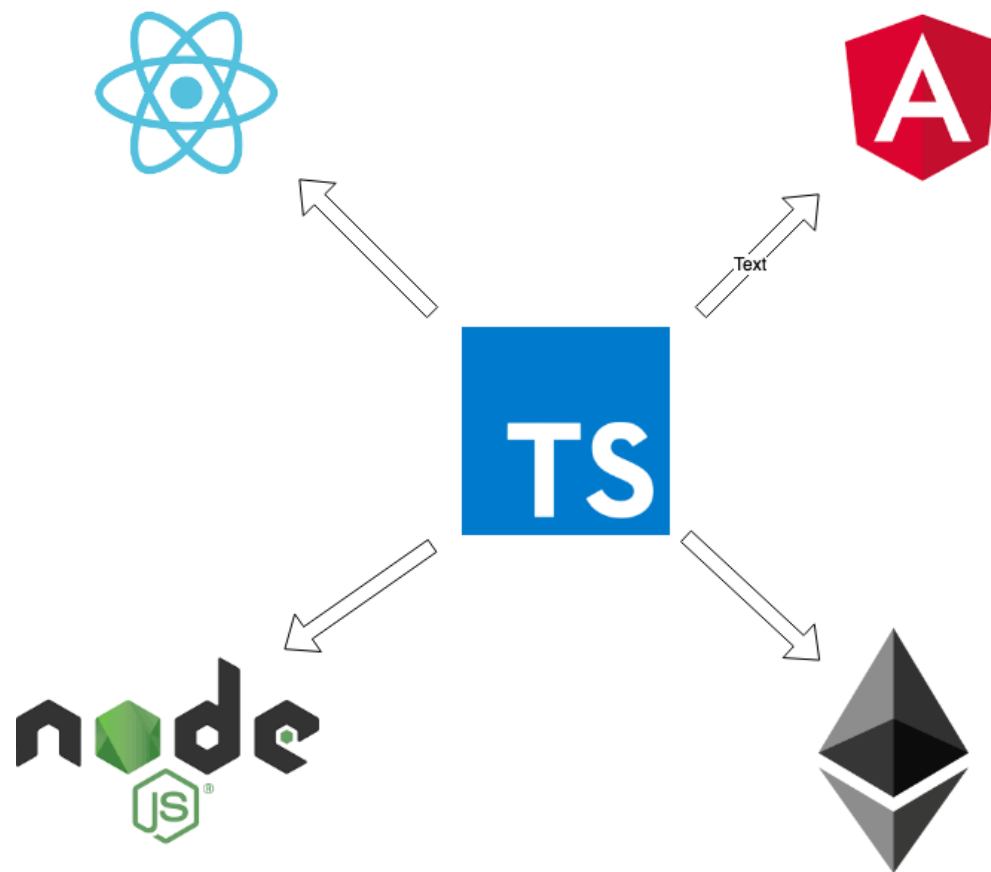
639 contributions in the last year

Contribution settings

	Aug	Sep	Oct	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug
Mon													
Wed													
Fri													

Learn how we count contributions.

LessMore



Chapter I

Migration

@ts-check

package.json

```
"type-check": "tsc src/main.js --allowJs --out  
/dev/null",
```

Migrate

```
for f in src/**/*.js; do  
    git mv "$f" "${f%.js}.ts"  
done
```

Chapter II

Compiler options

Dead code elimination

```
"noUnusedParameters": true,  
"noUnusedLocals": true,  
  
const fn = (_unusedArg) => 42 // OK
```

strictFunctionTypes: true,

```
const fn = (arg: number, arg1: (string) => string)  
=> {}
```

```
fn(42, str => str / 2); // ERROR
```

"noImplicitAny": true,

```
// ERROR
```

```
const fn = (arg) => arg;
```

```
// OK
```

```
const fn1 = (arg: any) => arg;
```

"strictNullChecks": true,

```
document.querySelector('#id').getAttribute('class');  
// ERROR;
```

```
const element = document.querySelector('#id');  
if (element !== null) {  
    element.getAttribute('id'); // OK  
}
```

```
(document.querySelector('#id') as  
HTMLElement).getAttribute('id'); // OK
```

Honorable mentions

```
"noImplicitReturns": true,  
"noImplicitThis": true,  
"strictBindCallApply": true,  
"paths": {  
  "@core/*": [  
    "app/*"  
  ],  
}
```

CHAPTER III

Cheap Tricks

Auto types

```
const fn = (): number =>
42;
const variable: string =
'';
public attr: boolean =
true;
[1, 2, 3].map((x: number)
=> x + 1);
const number$ = of<number>
(42);
```

```
const fn = () => 42;
const variable = '';
public attr = true;
[1, 2, 3].map(x => x + 1);
const number$ = of(42);
```

Property Accessing

```
interface Data {  
  field: {  
    name: string;  
  }  
}  
  
export const fn = (arg: Data['field']) => {  
  return arg.name; // { name: string }  
}  
  
export const fn1 = (name: Data['field']['name'])  
=> {  
  return name; // string  
}
```


Big integer

```
const bigNumber = 123_456_789;
```

Readonly

```
class Service extends AbstractService {  
  public readonly url; = '...';  
}
```

```
const service = new Service();  
service.field = '???'; // ERROR
```

Tuple and dict

```
const fn = (arg: { [key: string]: number }) => {  
  const val = arg.key1 + arg.key2 + arg.key3; //  
number  
  const val1 = arg.totallyRandomKey; // number  
  const val2 = arg['wpłynąłem na suchego przestwór  
oceanu']; // number  
};
```

```
let tuple: [string, number];  
tuple = ["hello", 10]; // OK  
tuple = [10, "hello"]; // Error  
let str = tuple[0]; // string  
let num = tuple[1]; // number  
let len = tuple.length // 2
```

PART IV
Zbiory

Merged types

```
const fn = (arg: { key: string } & { key1: number  
  }) => 42;
```

```
fn({ key: '42' }); // ERROR
```

```
fn({ key1: 42 }); // ERROR
```

```
fn({ key: '42', key1: 42 }); // GOOD
```

```
type tableRow = Item & { selected?: boolean };
```

Union types

```
const fn = (arg: string | number) => {  
  arg.split(''); // ERROR  
  arg / 2; // ERROR  
  arg + 1; // OK  
  if (typeof arg === "string") {  
    arg.split('');  
  }  
  if (typeof arg === "number") {  
    arg / 2;  
  }  
}
```

Unknown types

```
function fn(arg: unknown) {  
  if (typeof arg === "string" || typeof arg ===  
"number") {  
    arg; // string | number  
  }  
  if (x instanceof Error) {  
    arg; // Error  
  }  
  if (isFunction(x)) {  
    arg; // Function  
  }  
}
```

Never types

```
function error(x): never {  
    throw new Error("Unexpected object: " + x);  
}  
  
let variable = error(42); // never
```


Never types

```
function(arg: never) {  
    ...  
}
```

CHAPTER VI

Values as types

Based on argument

```
interface Data {  
  fn(arg: -1): never  
  fn(arg: 0): []  
  fn(arg: number): Array<number>  
}  
  
let data: Data;  
const a = data.fn(42); // Array<number>  
const c = data.fn(0); // [];  
const d = data.fn(-1); // never
```

Based on key

```
export interface API {  
  "/users": { params: [], response: IUser[] }  
  "/user/:id": { params: [number], response:  
IUser }  
}
```

CHAPTER VII

Weird parts

Constant

```
const someReduxAction = () => ({  
  type: ActionTypes.Some,  
}); // () => { type: type: ActionTypes }  
  
const otherReduxAction = () => ({  
  type: ActionTypes.Other,  
} as const); // () => { type: type:  
ActionTypes.Other }
```

Resolve Union

```
const some reducer(state, action: SomeAction |  
OtherAction) => {  
  if (action.type === ActionTypes.Some) {  
    action.payload; //SomeAction  
  }  
}
```

Generics extends

```
function loggingIdentity<T>(arg: T): T {  
    console.log(arg.length); // Error: T doesn't  
    have .length  
    return arg;  
}  
  
function loggingIdentity<T extends Array<any>>(arg:  
T): T[number] {  
    console.log(arg.length); // OK  
    return arg[0];  
}  
  
loggingIdentity(42) // ERROR  
loggingIdentity([]) // OK  
loggingIdentity([42]) // number
```


Keyof

```
interface API {  
    "/users": { params: [], response: User[] }  
    "/user/:id": { params: [number], response:  
User}  
}  
  
let fetch = <T extends keyof API>(api: API, url:  
T): API[T] => {  
    return api[url];  
}  
  
let r = fn({} as API, '/users'); // { params: [],  
response: User[]}  
let r1 = fn({} as API, '/uSers'); // ERROR
```

Optional extends

```
type If<A extends boolean, T, U> = A extends true  
? T : U;
```

```
let a: If<true, string, number>; // string  
let b: If<false, string, number>; // number
```

Optional extends

```
const first = (arr) => arr[0]
```

```
const first1 =
```

```
  <T extends any[]>(arr: T): T extends [] ? never  
: T[number] => arr[0];
```

```
let never = first1([]); // never
```

```
let num = first1([42]); // number
```

Infer

```
type ReturnType<T extends Function> =  
    T extends (...args: any[]) => infer R ? R :  
never;  
  
type R = ReturnType<() => 42> // 42
```

Maped types

```
type ReadonlyObject<A> = { readonly [K in keyof  
A]: <A[K]> };
```

```
type DeepReadonlyObject<A> = { readonly [K in  
keyof A]: DeepReadonly<A[K]> }
```

```
type X = DeepReadonlyObject<{ key: string, key1:  
number }>; // { readonly key: any; readonly key1:  
any; }
```

Maped types examples

```
type Optional<A extends object> = { [K in keyof  
A]?: <A[K]> };  
type Required<A extends object> = { [K in keyof  
A]-?: <A[K]> };}
```

Optional mapped types # json api

```
interface User {  
  id: string;  
  name: string;  
  age: number;  
  courses: Array<Course>  
}
```

```
interface UserResponse {  
  id: string;  
  attributes: {  
    name: string;  
    age: number  
  }  
  relationships { courses: { data: [] } }  
}
```

Examples

```
interface Response<T extends { id: string }, R =  
void> {  
    id: string;  
    type: string;  
    attributes: Omit<T, 'id'>;  
    relationships: R extends void ? void : { [key:  
string]: { data: Array<R> } };  
}  
  
type UserResponse = Response<User, Course>;
```


Grande finale

```
type EmptyTuple = [];
```

```
type TupleLength<T extends Array<any>> =  
T["length"];
```

```
type PrependTuple<A, T> = T extends Array<any>  
  ? (((a: A, ...b: T) => void) extends (...a:  
infer I) => void ? I : [])  
  : [];
```

```
type NumberToTuple<N extends number, L extends  
Array<any> = EmptyTuple> = {  
  true: L;  
  false: NumberToTuple<N, PrependTuple<1, L>>;  
}[TupleLength<L> extends N ? "true" : "false"];
```

Thank you :*