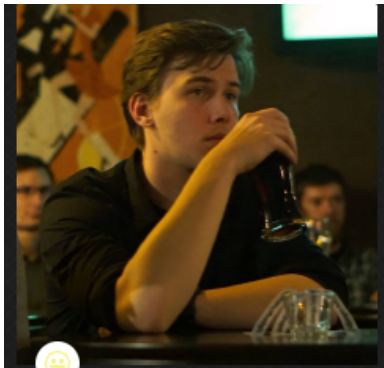
TYPESCRIPT: COLLECTED WORKS

https://github.io/przemyslawjanpietrzak

https://twitter.com/przemyslawjanp

https://stackoverflow.com/users/5914352/przemyslaw-pietrzak



Set your status

Przemyslaw Pietrzak

przemyslawjanpietrzak

PRO

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

Pinned repositories

Customize your pinned repositories

rembrandt

Simple functional UI framework written in Reasonml.

OCaml ★ 39

pyMonet

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

Python ★ 16

RxTowerDefense

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

TypeScript ★ 6 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.

Shell

Part I
Migration

Demo (migracja)

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
```

Part 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
```

"noImpicitAny": true,

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

// GOOD
const fn1 = (arg: any) => arg;

// GOOD
const fn2 = (arg: number) => arg;

// ALSO GOOD
[1,2,3].map(item => item + 1);
```

"strictNullChecks": true,

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

(document.querySelector('#id') as HTMLElement).getAttribute('class')

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

Honorable mentions

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

PART III
Tricks

Auto types

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

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

Property Accessing

```ts
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 & abstract

```
abstract class AbstractService {
  public method() {}
}

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

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

const abstractService = new AbstractService(); // 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
```

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(''); // Property 'split' does not exist on type 'string | number'.
    arg / 2; // The left-hand side of an arithmetic operation must be of type 'any', 'number', 'bigint' or an enum type.
    arg + 1; // OK
    if (typeof arg === "string") {
        return arg.split('');
    }
    if (typeof arg === "number") {
        return arg / 2;
    }
}
```

Unknown types

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

Never types

```
function error(x): never {
  throw new Error("Unexpected object: " + x);
}

let variable = error(42); // never
```

Never types

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

PART V
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}
}
```

PART VI
Weird parts

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
```

Generics extends

```
type User = Object;

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

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

let never = first1([]);
let int = first1([42]);
```

Infer

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

type R = ReturnType<() => 42> // 42
```

Infer

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

type Result = PrependTuple<42, [43, 44]> // [42, 43, 44]
```

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

```
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

```
export type Omit<A extends object, K extends keyof A> = Pick<A,
Exclude<keyof A, K>>

type X = Omit<{ key: string, key1: string }, "key"> //  { key1: string; }

type Diff<A extends object, K extends keyof A> = Omit<A, K> &
Partial<Pick<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: []}}
}
```

Optional mapped types # json api

```
interface Response<T extends object, 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"];

type Increment<N extends number> = TupleLength<PrependTuple<1,
NumberToTuple<N>>>;

type T = Increment<42>
```

Thank you :*