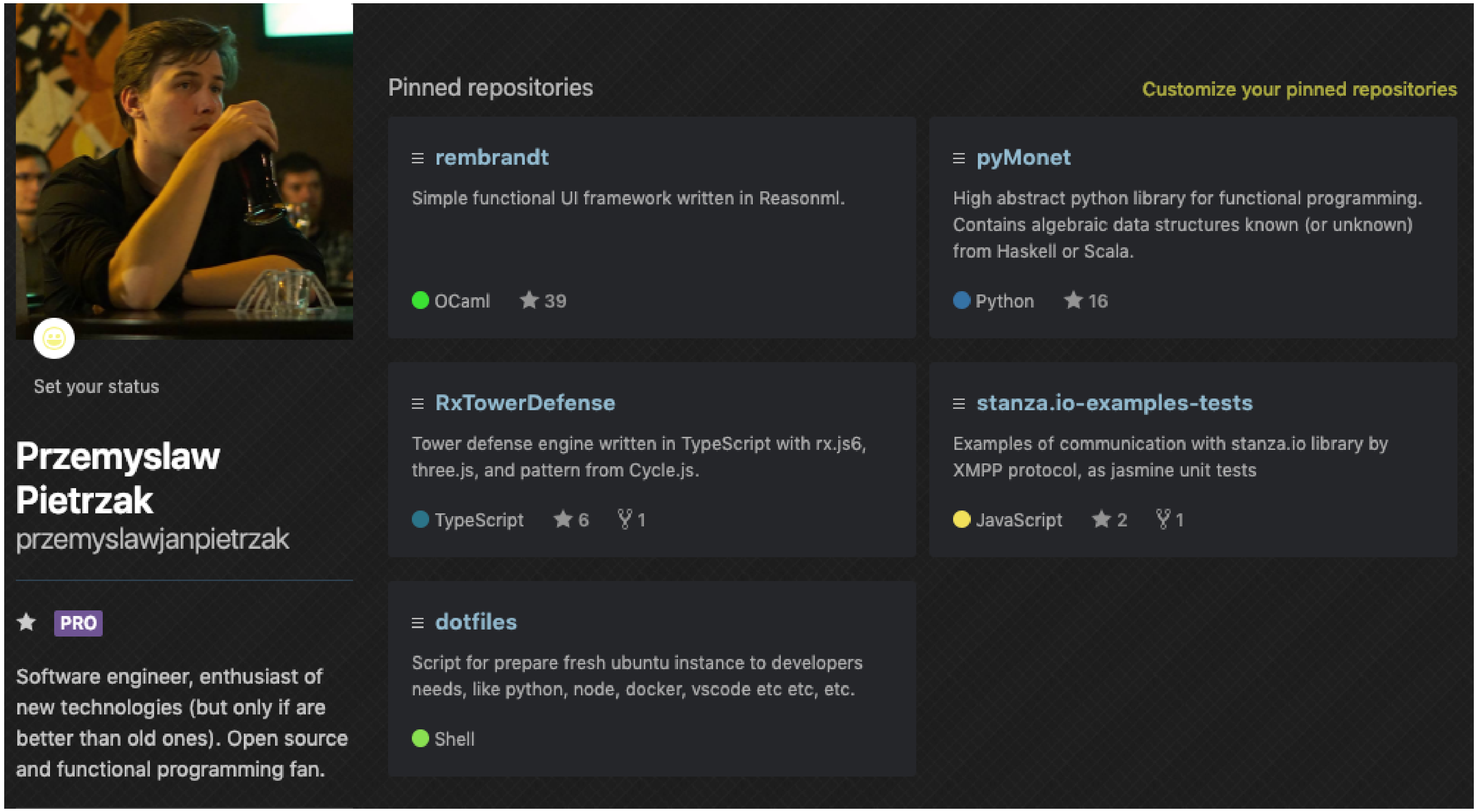
TYPESCRIPT FOR JAVA DEVELOPERS

https://github.com/przemyslawjanpietrzak



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

rembrandt
Simple functional UI framework written in Reasonml.
OCaml ★ 39

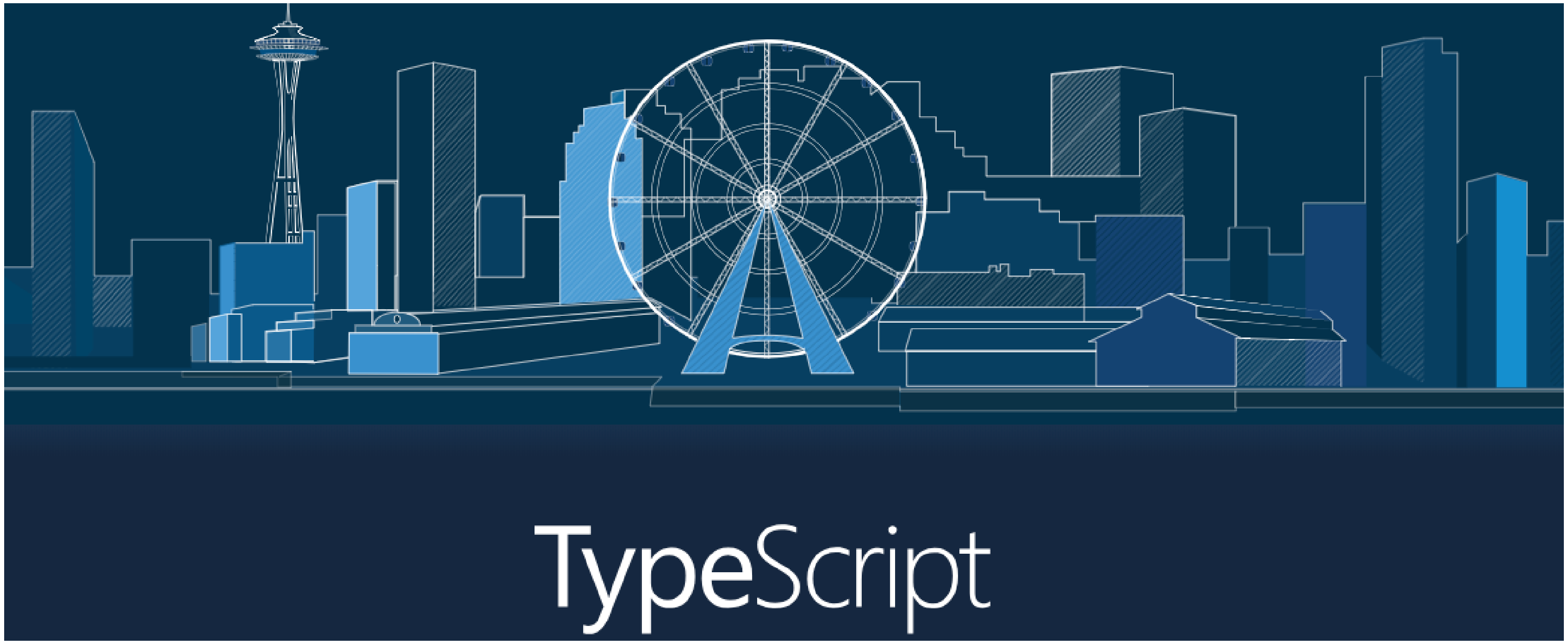
pyMonet
High abstract python library for functional programming. Contains algebraic data structures known (or unknown) from Haskell or Scala.
Python ★ 16

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

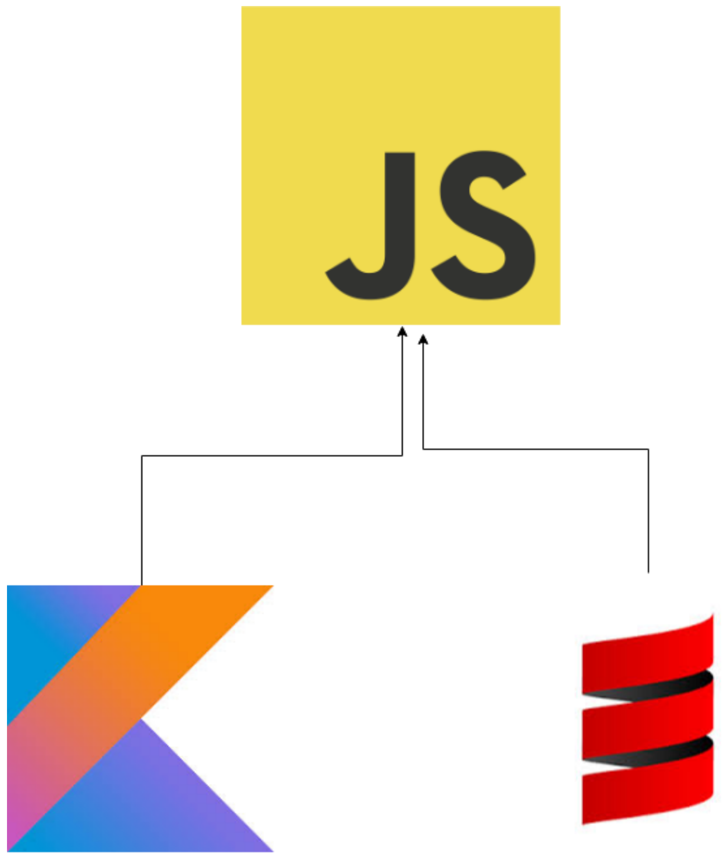
dotfiles
Script for prepare fresh ubuntu instance to developers needs, like python, node, docker, vscode etc etc, etc.
Shell

JavaScript: The Good Parts
Crockford
O'REILLY
JavaScript
The Definitive Guide



- JavaScript superset
- OOP & static typed language
- Without own runtime

WHAT ABOUT JVM LANGUAGES?



FRONTEND FRAMEWORKS

PROLOG

Types declarations

```
class Object {
  attribute: string;

  method(arg1: string, agr2: boolean): number {
    ...
  }
}
```

Access

```
class Object {
  private attribute: string;

  public method(arg1: string, agr2: boolean): number {
    ...
  }

  protected methdod1() {
    ...
  }
}
```

Abstract and final

```
abstract class Object {
  public readonly url = '127.0.0.1';
}
```

Generics

```
const fn = <T>(arg: T): Array<T> => [arg];
const variable = fn(3) // Array<number>;
```

Interfaces

```typescript
interface Developer {
  name: string;
  code(lang: string): string;
}

class Human implements Developer {
  name = 'John';
  code(lang: string) {
    return 'TypeScript';
  }
}
```

Enums

```typescript
enum Response {
  No = 0,
  Yes = 1,
}

const respond = (recipient: string, message: Response): void {
    // ...
}

respond("Princess Caroline", Response.Yes)
```

CHAPTER I
Migration

DEMO

package.json

```
tsc main.js --allowJs --checkJs --out /dev/null
```

MIGRATE

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

CHAPTER II
Differences

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: boolean = false;
[1, 2, 3].map(x => x + 1);
const number$ = observableOf(42);
```

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;
    }
}
```

Type aliases

```
type AliasType = Array<{ [key: string]: [number, string, boolean] }>
```

Any type

```
const fn = (explicite: any, implicite) => {
  explicite *= 42;
  implicite.map(i => i + 2);
  explicite.notExistingMethod();
  // no compilation error
}
```

PART III
Compiler options

Dead code elimination

tsconfig.json

```
"noUnusedParameters": true,
"noUnusedLocals": true,
```

```
const fn = (arg: number) => { // ERROR
 const variable = 42; // ERROR
 return null;
}

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

"strictFunctionTypes": true,

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

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

"noImplicitThis": true,

```
function fn() {
  console.log(this); // 'this' implicitly has type 'any' because it does not have a type annotation.
}

class Obj {
  method() {
    console.log(this); // OK
    function fn() {
      console.log(this); // 'this' implicitly has type 'any' because it does not have a type annotation.
    }
  }
}
```

"noImplicitReturns”: true

```
const fn = () => {
    if (true) {
        return; // ERROR: Not all code paths return a value.
    }
    return 42;
}
```

"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')
<HTMLElement>document.querySelector('#id').getAttribute('class')

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

Honorable mentions

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

PART IV
Tricks

Generics extends

```
function loggingIdentity<T>(arg: T): T {
    console.log(arg.length);  // Error: T doesn't have .length
    return arg;
}

interface Lengthwise {
    length: number;
}

function loggingIdentity<T extends Lengthwise>(arg: T): T {
    console.log(arg.length);  // Now we know it has a .length property, so no more
error
    return arg;
}

loggingIdentity({}) // ERROR
```

Conditional Generics

```typescript
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
```

Mapped types

```
export 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: number; }
```

Optional mapped types

```
export type Omit<A extends object, K extends string | number | symbol> = 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>>
```

Grande finale

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

type NumberToTuple<N extends number, L extends Array<any> = ZeroTuple> = {
  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 :*

Btw We're hiring!

#chopokodzic

espeo.eu