TSD

Test your typeS, stupiD

Types? Tests?

> We have tests and we have types

But do you have tests for your types?

Tests

totally not made up example

```
function addOne(a: number): number {
    return a + 1
}

test('adds one', () => {
    expect(addOne(1)).toEqual(2)
})
```

Tests

totally not made up example

```
function addOne(a: number): number {
    return a + 1
}

test('adds one', () => {
    expect(addOne(1)).toEqual(2)
})
```

Tests

totally not made up example

```
function addOne(a: number): number {
    return a + 1
}

test('adds one', () => {
    expect(addOne(1)).toEqual(2)
})
```

Tests - Without Type

```
function addOne(a: number | null): number {
    return a ?? 0 + 1
}

test('adds one', () => {
    expect(addOne(1)).toEqual(2)
})
```

Tests - Without Type

```
function addOne(a: number | null): number {
    return a ?? 0 + 1
}

test('adds one', () => {
    expect(addOne(1)).toEqual(2)
})
```

Tests - Without Type

```
function addOne(a: number | null): number {
    return a ?? 0 + 1
}

test('adds one', () => {
    expect(addOne(1)).toEqual(2)
})
```

Tests - Including the Type

```
function addOne(a: number | null): number {
    return a ?? 0 + 1
}

test('adds one', () => {
    const expectType: (a: number) => number = addOne

    expect(addOne(1)).toEqual(2)
})
```

Tests - Including the Type

```
function addOne(a: number | null): number {
    return a ?? 0 + 1
}

test('adds one', () => {
    const expectType: (a: number) => number = addOne

    expect(addOne(1)).toEqual(2)
})
```

Tests - Including the Type

```
function addOne(a: number | null): number {
    return a ?? 0 + 1
}

test('adds one', () => {
    const expectType: (a: number) => number = addOne

    expect(addOne(1)).toEqual(2)
})
```

Excourse: Subtyping

```
const addOneV0 = (a: number) =>  number
const addOneV1 = (a: number | null): number

type A = number | null
type SubTypeOfA = number

let a: A
let subTypeOfA: SubTypeOfA

a = subTypeOfA // works: number | null = number

subTypeOfA = a // noooo: number = number | null
```

Excourse: Subtyping

```
const addOneV0 = (a: number) =>  number
const addOneV1 = (a: number | null): number

type A = number | null
type SubTypeOfA = number

let a: A
let subTypeOfA: SubTypeOfA

a = subTypeOfA // works: number | null = number

subTypeOfA = a // noooo: number = number | null
```

Excourse: Subtyping

```
const addOneV0 = (a: number) =>  number
const addOneV1 = (a: number | null): number

type A = number | null
type SubTypeOfA = number

let a: A
let subTypeOfA: SubTypeOfA

a = subTypeOfA // works: number | null = number

subTypeOfA = a // noooo: number = number | null
```

Excourse: Subtyping

```
const addOneV0 = (a: number) =>  number
const addOneV1 = (a: number | null): number

type A = number | null
type SubTypeOfA = number

let a: A
let subTypeOfA: SubTypeOfA

a = subTypeOfA // works: number | null = number

subTypeOfA = a // noooo: number = number | null
```

Excourse: Subtyping

```
const addOneV0 = (a: number) =>  number
const addOneV1 = (a: number | null): number

type A = number | null
type SubTypeOfA = number

let a: A
let subTypeOfA: SubTypeOfA

a = subTypeOfA // works: number | null = number

subTypeOfA = a // noooo: number = number | null
```

Tests - Including the Type

```
function addOne(a: number | null): number {
    return a ?? 0 + 1
}

test('adds one', () => {
    const expectType: (a: number) => number = addOne

    expect(addOne(1)).toEqual(2)
})
```

Tests - Including the Type

```
function addOne(a: number | null): number {
    return a ?? 0 + 1
}

test('adds one', () => {
    const expectType: (a: number) => number = addOne

    expect(addOne(1)).toEqual(2)
})
```

Tests - Including the Type

```
function addOne(a: number | null): number {
    return a ?? 0 + 1
}

test('adds one', () => {
    const expectType: (a: number) => number = addOne

    expect(addOne(1)).toEqual(2)
})
```

Tests - Including the exact Type

```
function addOne(a: number | null): number {
    return a ?? 0 + 1
}

test('adds one', () => {
    const expectType0: (a: number) => number = addOne
    const expectType1: typeof addOne = (a: number) => number

    expect(addOne(1)).toEqual(2)
})
```

Tests - Including the exact Type

```
function addOne(a: number | null): number {
    return a ?? 0 + 1
}

test('adds one', () => {
    const expectType0: (a: number) => number = addOne
    const expectType1: typeof addOne = (a: number) => number

    expect(addOne(1)).toEqual(2)
})
```

Tests - Including the exact Type

```
function addOne(a: number | null): number {
    return a ?? 0 + 1
}

test('adds one', () => {
    const expectType0: (a: number) => number = addOne
    const expectType1: typeof addOne = (a: number) => number

    expect(addOne(1)).toEqual(2)
})
```

Tests - Recap

- we write *code* with **types**
- we test the *code*
- we test the **types**

That Same Test with TSD

``npm install tsd`` a couple of gigabytes and some package.json configuration later

```
import { expectAssignable, expectError, expectType } from "tsd"
import { addOne } from '../src'

expectType<(a:number) => number>(addOne)
expectError(addOne(null))
expectError(addOne(undefined))
```

```
$> node_modules/.bin/tsd


  test-d/addOne.test-d.ts:5:0
  ✖  5:0  Parameter type (a: number) => number is declared too wide for
argument type (a: number | null) => number.
  ✖  6:0  Expected an error, but found none.

  2 errors
```

Thanks for listening

Haha not yet

Haha not yet

- that example above was wa*aaa*aay to simple

Haha not yet

- that example above was wa*aaa*aay to simple
- real world types look more like:

```
// ts builtin utility
type Partial<T> = {
    [P in keyof T]?: T[P];
}
```

Real world type test

```
interface Todo {
    title: string
    description :string
}

declare const partial: Partial<Todo>

expectType<{title?: string, description?: string}>(partial)
```

Real world type test 2 - modifying the Partial type

```
type PartialPrefix<T, P extends string> = {
  [K in keyof T as (K extends string ? `${P}${K}` : never)]?: T[K];
};
```

Real world type test 2 - modifying the Partial type

```
interface Todo {
    title: string
    description :string
}

declare const partialPrefix: PartialPrefix<Todo, 'initial' | 'final'>

expectType<{
  initial_title?: string
  final_title?: string
  initial_description?: string
  final_description?: string
}>(partialPrefix)
```

Cheers

TypeFest uses TSD (much code):

https://github.com/sindresorhus/type-fest/tree/main/test-d