
TS
JS

TypeScript

Static typechecker for JavaScript programs

staticTypechecker/greet.ts

```ts
type Person = string;

type Greet = (person: Person, date: Date) => void;

const greet: Greet = (person, date) => {
  console.log(`Hello ${person}, today is ${date}!`);
};

greet('Luke');
```

```
$ npm i -g typescript
```

```
$ tsc --noEmitOnError staticTypechecker/greet.ts
greet.ts:9:1 - error TS2554: Expected 2 arguments, but got 1.

9 greet("Luke");
  ~~~~~~~~~~~~~~~

  greet.ts:3:31
    3 type Greet = (person: Person, date: Date) => void;
                                    ~~~~~~~~~~
    An argument for 'date' was not provided.


Found 1 error in greet.ts:9
```

staticTypechecker/betterGreet.ts

```
...

greet('Luke', new Date());
```

```
$ tsc --noEmitOnError staticTypechecker/betterGreet.ts
```

staticTypechecker/betterGreet.js

```
var greet = function (person, date) {
    console.log("Hello ".concat(person, ", today is ").concat(date, "!"));
};
greet("Luke", new Date());
```

Types

Basic Types

types/basicTypes.ts

```ts
type Primitive = string | number | boolean | bigint | symbol; // union
type Person = {
  name: Primitive;
  age: Primitive;
  isAwesome?: Primitive; // optional
};

const luke: Person = {
  name: 'Luke',
  age: 25, // at heart
  // isAwesome: ??? // undefined, jury's out
};
```

Union Types

types/unionTypes.ts

```ts
type Flag = 'auto' | 'all' | 'rec'; // literal union
type Options = { width: number; height: number };

const configure = (x: Options | Flag) => {};

configure({ width: 100, height: 100 });
configure('auto');
configure('automatic');
// Argument of type '"automatic"' is not
//   assignable to parameter of type 'Options | Flag'
```

Function Types

Function Type Expressions

types/functionTypes/functionTypeExpressions.ts

```ts
const greeter = (fn: (a: string) => void) => fn('Hello, World');

greeter(console.log);

// Parameter name is required:
type HiFn = (string) => void;
// Parameter has a name but no type.
//   Did you mean 'arg0: string'?
// (parameter) string: any
```

Generic Functions

types/functionTypes/genericFunctions.ts

```ts
// const firstElement: (arr: any[]) => any
const firstElement = (arr: any[]) => arr[0];

// const firstElementToo: <Type>(arr: Type[]) => Type | undefined
const firstElementToo = <Type>(arr: Type[]): Type | undefined => arr[0];

// const s: string | undefined
const s = firstElementToo(['a', 'b', 'c']);
// const u: undefined
const u = firstElementToo([]);

// Inference

const map = <Input, Output>(arr: Input[], func: (arg: Input) => Output): Output[] => arr.map(func);

// (parameter) n: string
// const parsed: number[]
const parsed = map(['1', '2', '3'], (n) => parseInt(n));
```

Constraints

types/functionTypes/constraints.ts

```ts
const longest = <T extends { length: number }>(a: T, b: T) =>
  (a.length >= b.length) ? a : b;

// const longerArray: number[]
const longerArray = longest([1, 2], [1, 2, 3]);
// const longerString: "alice" | "bob"
const longerString = longest('alice', 'bob');

const notOK = longest(10, 100); // ❌
// Argument of type 'number' is not assignable
//   to parameter of type '{ length: number; }'
```

Constraints (cont'd)

types/functionTypes/constraintsContd.ts

```ts
type MinimumLength = <T extends { length: number }>(obj: T, min: number) => T;

const minLength: MinimumLength = (obj, min) => {
  if (obj.length >= min) {
    return obj;
  } else {
    return { length: min };
// T '{ length: number; }' is not assignable to type 'T'.
//   '{ length: number; }' is assignable to the constraint of type 'T',
//     but 'T' could be instantiated with a different
//     subtype of constraint '{ length: number; }'.
  }
};

// 'arr' gets value { length: 6 }
const arr = minimumLength([1, 2, 3], 6);
// and crashes here because the returned object does not have a 'slice' method!
console.log(arr.slice(0));
```

Specifying Type Arguments

types/functionTypes/specifyingTypeArguments.ts

```
const combine = <T>(a1: T[], a2: T[]): T[] => a1.concat(a2);

const arr = combine([1, 2, 3], ["hello"]);
// Type 'string' is not assignable to type 'number'.

const shhIsOk = combine<string | number>([1, 2, 3], ["hello"]);
```

Guidelines

types/functionTypes/guidelines.ts

```ts
// Push type parameters down
const firstElement1 = <T>(a: T[])  => a[0];
const firstElement2 = <T extends any[]>(a: T)  => a[0];
const a = firstElement1([1, 2, 3]); // ✅ const a: number
const b = firstElement2([1, 2, 3]); // ❌ const b: any

// Use fewer type Parameters
const filter1 = <T>(a: T[], f: (arg: T) => boolean): T[] => a.filter(f); // ✅
const filter2 = <T, F extends (arg: T) => boolean>(a: T[], f: F): T[] => a.filter(f); // ❌

// Type parameters should appear twice
const greet1 = <Str extends string>(s: Str) => console.log('Hello, ' + s); // ❌
const greet2 = (s: string) => console.log('Hello, ' + s); // ✅
```

Structural Typing

differences/structuralTyping.ts

```ts
type Ball = { r: number };
type Sphere = { r: number };
type Tube = { r: number; l: number };

let ball: Ball = { r: 10 };
let sphere: Sphere = { r: 20 };
let tube: Tube = { r: 12, l: 3 };

// ball = sphere; sphere = ball; ball = tube; sphere = tube; // ✅
// tube = ball; tube = sphere; // ❌

let createBall = (r: number) => ({ r });
let createSphere = (r: number, metric: boolean) => ({ r: metric ? r : r * 0.39 });
let createRedBall = (r: number) => ({ r, color: 'red' });

// createSphere = createBall; createBall = createRedBall; // ✅
// createBall = createSphere; // ❌
// createRedBall = createBall; // ❌
```

Nominal Typing

differences/nominalTyping.ts

```ts
type USD = number & { __brand: 'USD' };
type Brand<K, T> = K & { __brand: T };
type EUR = Brand<number, 'EUR'>;

const usd = 10 as USD; // `as`
const eur = 10 as EUR; // `as`

const gross = (net: USD, tax: USD): USD => (net + tax) as USD;

gross(usd, usd); // ✅
gross(eur, usd); // ❌
// Argument of type 'EUR' is not assignable to parameter of type 'USD'.
  // Type 'EUR' is not assignable to type '{ __brand: "USD"; }'.
    // Types of property '__brand' are incompatible.
      // Type '"EUR"' is not assignable to type '"USD"'.
```

Interface vs Type

differences/interface.ts

```ts
interface Animal {
  name: string;
}

interface Bear extends Animal {
  honey: boolean;
}

const bear = {} as Bear;
bear.name;
bear.honey;

interface Bear extends Animal {
  kind: 'grizzly' | 'black' | 'polar';
} // reopen Bear
```

differences/typeAlias.ts

```ts
type Animal = {
  name: string;
};

type Bear = Animal & {
  honey: boolean;
};

const bear = {} as Bear;
bear.name;
bear.honey;

type Bear = Animal & {
  kind: 'grizzly' | 'black' | 'polar';
}; // Error: Duplicate identifier Bear
```