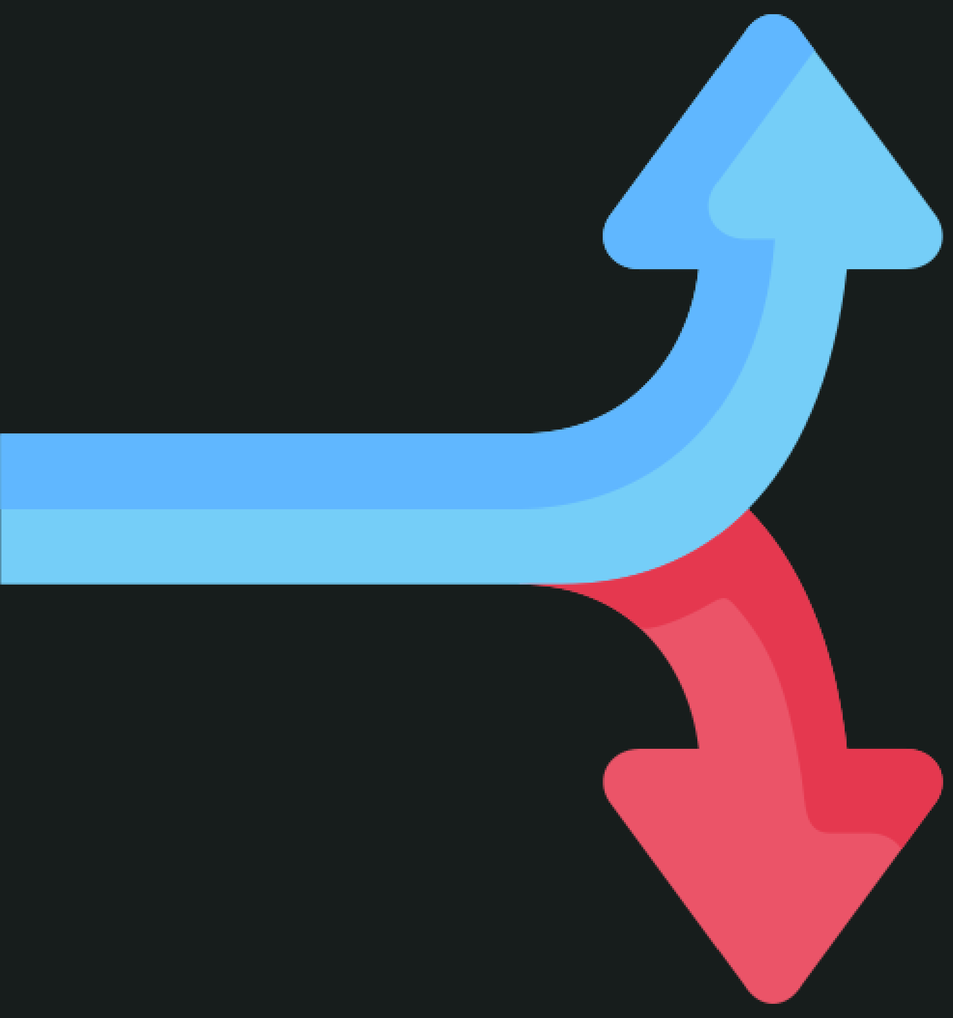
Gagan Saini
@gagan-saini-gs

Github: Gagan-Saini-GS

Interface



Why Different?

Type

Gagan Saini
@gagan-saini-gs

Github: Gagan-Saini-GS

Interface

Object Shape Definition

They are primarily used for **defining the shape** of objects.

They allow you to **specify the structure** of an object, including its properties and their types.

Compatibility

It’s declarations are **open-ended**, meaning that you can **add new** properties to an interface later without causing **compatibility** issues with existing code that uses that interface.

Gagan Saini
@gagan-saini-gs

Github: Gagan-Saini-GS

Declaration Merging

It support declaration merging, meaning that **multiple interface** declarations with the same name are **merged into a single** interface.

This allows you to **extend the definition** of an interface across **multiple declarations**.

Extensibility

It support **extending other interfaces**, allowing you to create more complex and reusable type definitions by **combining multiple** interfaces.

Code Example

interface.tsx

```tsx
interface Person {
  name: string;
  age: number;
}

interface Employee extends Person {
  id: number;
}
```


Gagan Saini
@gagan-saini-gs

Github: Gagan-Saini-GS

Type

Flexible Shape Definition

Type aliases allow you to **define any kind of type**, not just objects.

They can be used to **create aliases** for primitive types, union types, intersection types, & more.

Union and Intersection Types

Type aliases support **defining union types** and **intersection types**, allowing you to combine multiple types into a single type.

Expressiveness

Type aliases are **more expressive** than **interfaces** in some cases, especially when dealing with complex types or conditional types.

Readability

Type aliases can be **more concise** and **easier to read** than interfaces, particularly for complex types or when **defining types inline**.

Code Example

type.tsx

```tsx
type Person = {
  name: string;
  age: number;
};

type Employee = Person & {
  id: number;
};
```

Summary

Interfaces are typically **used for defining the shape of objects**, supporting extension and declaration merging,

Type are more **versatile** and can be used to **define a wide range of types**, including objects, unions, intersections, and more.

The choice between **interfaces** and **type** aliases depends on the **specific requirements** of your code and **personal preference**.

Gagan Saini
@gagan-saini-gs

Github: Gagan-Saini-GS

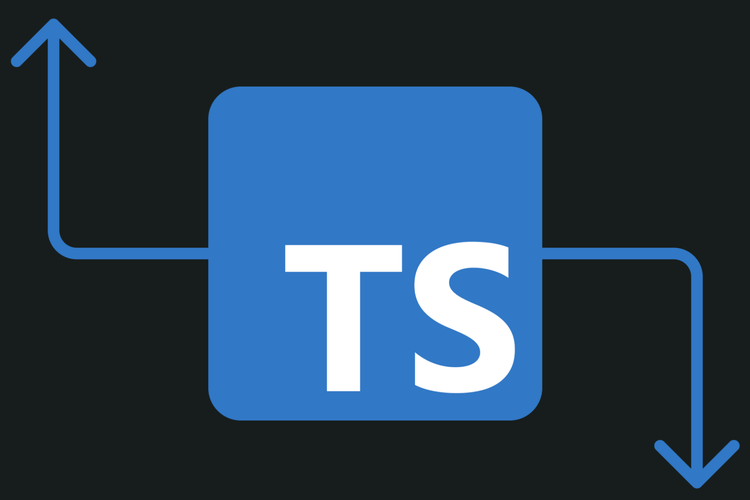
If found it useful then, Like & Share 💙

Learn What is TypeScript & it's Features?

Gagan Saini
@gagan-saini-gs

Github: Gagan-Saini-GS

What is TypeScript?



Why do you need this?

Gagan Saini
@gagan-saini-gs

Github: Gagan-Saini-GS

TypeScript

It is an **open-source** language developed and maintained by **Microsoft**.

It is a statically **typed superset of JavaScript** that adds optional static typing, classes, interfaces, and other features to the language.

TypeScript code is **transpiled into plain JavaScript** that can run on any browser or JavaScript engine.

Gagan Saini
@gagan-saini-gs

Features

Static Typing

It means developers can **exp types** of variables, paramete values in their code.

It helps **catch errors at comp** than runtime, improving code reducing bugs.

By providing type annotation more **self-documenting** & ea

Thanks for reading
Comment Feedback Below