



Education
@topdev_media



JS

JavaScript

vs

TypeScript

T

```
var name = "Alice"; // Can be redeclared  
let age = 25; // Block scope, cannot be redeclared  
const city = "Paris"; // Block scope, constant
```

JavaScript allows **flexible variable declarations** with var, let, and const.

VS

TypeScript **enhances variable declarations** by requiring **type annotations**, improving code reliability.

```
let name: string = "Alice";  
let age: number = 25;  
const city: string = "Paris";
```

```
function add(x, y) {  
  return x + y;  
}
```

JavaScript functions do not specify types, which can lead to **unexpected behaviors**.

VS

TypeScript **requires types** for function **parameters** and **return value**, ensuring values are used correctly.

```
function add(x: number, y: number): number {  
  return x + y;  
}
```

```
class Person {  
  constructor(name) {  
    this.name = name;  
  }  
  
  greet() {  
    return "Hello, " + this.name;  
  }  
}
```

JavaScript supports classes with ES6, enabling structured and **object-oriented** programming.

VS

TypeScript adds **access modifiers** and **types** to classes, which increase **encapsulation** and maintainability.

```
class Person {  
  private name: string;  
  
  constructor(name: string) {  
    this.name = name;  
  }  
  
  greet(): string {  
    return "Hello, " + this.name;  
  }  
}
```

```
function greet(person) {  
  return "Hello, " + person.name;  
}
```

No direct equivalent for interfaces in pure JavaScript.

VS

TypeScript's interfaces define **contracts** for **objects**, enhancing **code validation** and editor support.

```
interface Person {  
  name: string;  
  age: number;  
}  
  
function greet(person: Person): string {  
  return "Hello, " + person.name;  
}
```

```
// Errors can only be caught at runtime  
console.log(nonExistentVariable); // ReferenceError at runtime
```

JavaScript errors are often only **caught at runtime**, which can lead to less predictable code.

VS

TypeScript identifies **issues at compile-time**, like unassigned variables, preventing runtime errors.

```
let someVar: number;  
console.log(someVar);  
// Compilation error: variable used before being assigned
```




JavaScript does not have native support for generics.

VS

TypeScript's generics allow functions and classes to **operate with any data type**, not limited to one.

```
function identity<T>(arg: T): T {  
    return arg;  
}  
  
let output = identity<string>("myString");
```

```
// ES6 module syntax
export function greet(name) {
  return `Hello, ${name}`;
}
import { greet } from './greet';
```

JavaScript modules **improve code organization** and reuse by allowing separation into different files.

VS

TypeScript namespaces **encapsulate classes** and **functions**, preventing global namespace pollution.

```
namespace Greeting {
  export function sayHello(name: string) {
    return `Hello, ${name}`;
  }
}

let greeting = Greeting.sayHello("Alice");
```




JavaScript does not natively support decorators.

VS

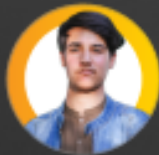
TypeScript decorators provide a way to **add annotations** and a **meta-programming syntax** for class declarations and members.

```
function sealed(constructor: Function) {
    Object.seal(constructor);
    Object.seal(constructor.prototype);
}

@sealed
class Greeter {
    greeting: string;
    constructor(message: string) {
        this.greeting = message;
    }
    greet() {
        return "Hello, " + this.greeting;
    }
}
```



Follow for more



in

Halim Shams