Table of Contents

[1. Overview of TypeScript 2](#_Toc200011153)

[**What is TypeScript?** 2](#_Toc200011154)

[**Example:** 2](#_Toc200011155)

[2. Environment Setup 2](#_Toc200011156)

[**Steps:** 2](#_Toc200011157)

[3. Basic Types 2](#_Toc200011158)

[**List of Basic Types with Examples:** 2](#_Toc200011159)

[4. Type Inference 3](#_Toc200011160)

[**Explanation:** 3](#_Toc200011161)

[5. Type Annotations 3](#_Toc200011162)

[**Explanation:** 3](#_Toc200011163)

[6. Functions and Parameter Types 3](#_Toc200011164)

[**Optional Parameters:** 3](#_Toc200011165)

[7. Arrays and Tuples 3](#_Toc200011166)

[**Arrays:** 3](#_Toc200011167)

[**Tuples:** 3](#_Toc200011168)

[8. Enums 4](#_Toc200011169)

[**Explanation:** 4](#_Toc200011170)

[9. Type Assertions 4](#_Toc200011171)

[**Explanation:** 4](#_Toc200011172)

[10. Literal Types 4](#_Toc200011173)

[**Explanation:** 4](#_Toc200011174)

[11. Union Types 4](#_Toc200011175)

[**Explanation:** 4](#_Toc200011176)

[12. Intersection Types 5](#_Toc200011177)

[**Explanation:** 5](#_Toc200011178)

[13. Type Aliases 5](#_Toc200011179)

[**Explanation:** 5](#_Toc200011180)

[14. Optional and Default Parameters 5](#_Toc200011181)

[15. Rest Parameters 5](#_Toc200011182)

[16. Nullable Types 6](#_Toc200011183)

[17. Interfaces 6](#_Toc200011184)

[18. Classes and Inheritance 6](#_Toc200011185)

[19. Access Modifiers (public, private, protected, readonly) 7](#_Toc200011186)

[20. Abstract Classes 7](#_Toc200011187)

[21. Static Members 8](#_Toc200011188)

**1. Overview of TypeScript**

**What is TypeScript?**

TypeScript is a superset of JavaScript that adds **static typing**, **interfaces**, and **compile-time error checking**.

* Developed by Microsoft.
* Helps catch errors early in development.
* Compiles to plain JavaScript.

**Example:**

let message: string = 'Hello, TypeScript!';

console.log(message);

**2. Environment Setup**

**Steps:**

1. Install Node.js from <https://nodejs.org>
2. Install TypeScript globally:

npm install -g typescript

1. Create a .ts file and compile:

tsc hello.ts

node hello.js

**3. Basic Types**

**List of Basic Types with Examples:**

|  |  |
| --- | --- |
| **Type** | **Example** |
| **string** | let name: string = 'Alice'; |
| **number** | let age: number = 30; |
| **boolean** | let isActive: boolean = true; |
| **any** | let data: any = 'Hello'; data = 42; |
| **unknown** | let value: unknown = 10; |
| **void** | Used in functions that return nothing |
| **never** | For functions that never return |
| **null** | let n: null = null; |
| **undefined** | let u: undefined = undefined; |
| **bigint** | let big: bigint = 9007199254740991n; |
| **symbol** | let sym = Symbol('desc'); |
| **object** | let obj: object = { key: 'value' }; |

**4. Type Inference**

**Explanation:**

TypeScript can automatically infer the type of a variable based on its value.

let greeting = 'Hi'; // inferred as string

// greeting = 123; // ❌ Error

**5. Type Annotations**

**Explanation:**

Explicitly defining the type of a variable, parameter, or return value.

let count: number = 10;

function greet(name: string): string {

return 'Hello, ' + name;

}

**6. Functions and Parameter Types**

function add(a: number, b: number): number {

return a + b;

}

**Optional Parameters:**

function greet(name?: string) {

console.log('Hello ' + (name || 'Guest'));

}

**7. Arrays and Tuples**

**Arrays:**

let numbers: number[] = [1, 2, 3];

**Tuples:**

let person: [string, number] = ['Alice', 25];

**8. Enums**

**Explanation:**

Enums allow a developer to define a set of named constants.

enum Direction {

Up,

Down,

Left,

Right

}

let move: Direction = Direction.Up;

**9. Type Assertions**

**Explanation:**

Used to tell the compiler to treat a variable as a specific type.

let someValue: any = 'Hello World';

let strLength: number = (someValue as string).length;

**10. Literal Types**

**Explanation:**

Restrict a variable to exact values.

let direction: 'left' | 'right';

direction = 'left'; // ✅

direction = 'up'; // ❌ Error

**11. Union Types**

**Explanation:**

A variable can hold multiple types.

let value: string | number;

value = 'Hello';

value = 100;

**12. Intersection Types**

**Explanation:**

Combines multiple types into one.

interface Name {

name: string;

}

interface Age {

age: number;

}

type Person = Name & Age;

const user: Person = { name: 'John', age: 30 };

## 13. Type Aliases

**Explanation:**

Gives a new name to a type.

type ID = string | number;

let userId: ID = 101;

## 14. Optional and Default Parameters

**Optional Parameters** are parameters that may or may not be passed. They are marked using ?.

function greet(name?: string) {

console.log(`Hello ${name ? name : "Guest"}`);

}

greet(); // Hello Guest

greet("Gaurab"); // Hello Gaurab

**Default Parameters** provide a default value if the parameter is not provided.

function greetUser(name: string = "Guest") {

console.log(`Welcome, ${name}`);

}

greetUser(); // Welcome, Guest

greetUser("Neha"); // Welcome, Neha

## 15. Rest Parameters

Used when a function accepts multiple arguments. It returns them as an array.

function sumAll(...numbers: number[]): number {

return numbers.reduce((sum, num) => sum + num, 0);

}

console.log(sumAll(1, 2, 3)); // 6

console.log(sumAll(5, 10, 15, 20)); // 50

## 16. Nullable Types

TypeScript has strict null checking. You can explicitly allow null or undefined.

let name: string | null = null;

name = "Gaurab";

function printLength(str: string | null) {

if (str) {

console.log(str.length);

} else {

console.log("No string provided.");

}

}

printLength(name); // 6

printLength(null); // No string provided.

## 17. Interfaces

Interfaces define the structure of an object, like a contract.

interface User {

name: string;

age: number;

isActive?: boolean;

}

let user1: User = {

name: "John",

age: 30,

};

console.log(user1);

## 18. Classes and Inheritance

**Class** defines a blueprint. **Inheritance** lets one class extend another.

class Person {

name: string;

constructor(name: string) {

this.name = name;

}

greet(): void {

console.log(`Hello, my name is ${this.name}`);

}

}

class Student extends Person {

rollNo: number;

constructor(name: string, rollNo: number) {

super(name);

this.rollNo = rollNo;

}

display(): void {

console.log(`${this.name} - Roll No: ${this.rollNo}`);

}

}

const student1 = new Student("Rahul", 101);

student1.greet(); // Hello, my name is Rahul

student1.display(); // Rahul - Roll No: 101

## 19. Access Modifiers (public, private, protected, readonly)

* public – accessible everywhere (default)
* private – accessible only inside the class
* protected – accessible in class and subclasses
* readonly – value can be set once

class Car {

readonly brand: string;

private speed: number = 0;

constructor(brand: string) {

this.brand = brand;

}

accelerate(value: number): void {

this.speed += value;

console.log(`${this.brand} speed: ${this.speed}`);

}

getSpeed(): number {

return this.speed;

}

}

const car = new Car("Toyota");

car.accelerate(50); // Toyota speed: 50

console.log(car.getSpeed()); // 50

## 20. Abstract Classes

Abstract classes cannot be instantiated directly. They provide a base for other classes to extend.

abstract class Animal {

constructor(public name: string) {}

abstract speak(): void;

move(): void {

console.log(`${this.name} is moving`);

}

}

class Dog extends Animal {

speak(): void {

console.log(`${this.name} says Woof!`);

}

}

const dog = new Dog("Tommy");

dog.speak(); // Tommy says Woof!

dog.move(); // Tommy is moving

## 21. Static Members

Static properties and methods belong to the class, not instances.

class MathUtil {

static PI = 3.1416;

static areaOfCircle(radius: number): number {

return this.PI \* radius \* radius;

}

}

console.log(MathUtil.PI); // 3.1416

console.log(MathUtil.areaOfCircle(5)); // 78.54