

Experiment – 1 a: TypeScript

Name of Student	Swaraj Patil
Class Roll No	39
D.O.P.	
D.O.S.	
Sign and Grade	

❖ Aim:

Write a simple TypeScript program using basic data types (number, string, boolean) and operators.

❖ Problem Statement:

- Create a calculator in TypeScript that uses basic operations like addition, subtraction, multiplication, and division. It also gracefully handles invalid operations and division by zero..
- Design a Student Result database management system using TypeScript.

❖ Theory :-

1. What are the different data types in TypeScript? What are Type Annotations in Typescript?

Number	number	Represents both integer and floating-point numbers.
String	string	Represents textual data.
Boolean	boolean	Represents logical values: true or false.
Null	null	Represents the intentional absence of any object value.
Undefined	undefined	Represents an uninitialized variable.
Symbol	symbol	Represents a unique, immutable value, often used as object keys.
BigInt	bigint	Represents integers with arbitrary precision.

2. How Do You Compile TypeScript Files?

TypeScript files (.ts) need to be compiled into JavaScript using the TypeScript compiler (tsc).

Steps:

Install TypeScript globally if not already installed:

npm install -g typescript

Compile a TypeScript file:

tsc filename.ts

Run the compiled JavaScript file:

node filename.js

To watch for changes and recompile automatically:

tsc filename.ts --watch

For larger projects, a tsconfig.json file can be used:

tsc --init

tsc

3. Difference Between JavaScript and TypeScript

Feature	JavaScript	TypeScript
Type System	Dynamically typed	Statically typed
Compilation	Directly runs in browsers	Needs compilation to JavaScript
Error Detection	Errors detected at runtime	Errors detected at compile-time
ES6 Features	Supports modern ES6+	Supports ES6+ and additional features like interfaces, generics
Code Readability	Can become complex in large projects	More maintainable and readable due to type safety
Performance	Slightly faster (no compilation)	Safer code, but needs compilation

4. What is the difference between Classes and Interfaces in Typescript? Where are interfaces used?

Feature	Class	Interface
Definition	Defines a blueprint for objects, including properties and methods.	Defines the structure of an object, specifying properties and method signatures without implementation.
Implementation	Can have constructors and method definitions.	Only declares method signatures and properties, without implementation.
Object Creation	Can be used to create objects using the <code>new</code> keyword.	Cannot be instantiated directly.
Inheritance	Supports inheritance using <code>extends</code> (single or multiple class inheritance).	Supports multiple inheritance using <code>extends</code> or can be implemented using <code>implements</code> .
Modifiers	Supports <code>public</code> , <code>private</code> , <code>protected</code> , and <code>readonly</code> .	Does not support access modifiers, all properties are public by default.
Usage	Used for creating objects with behavior (methods).	Used for enforcing structure and type checking.

Example of an Interface in TypeScript

typescript

```
interface Animal {
    name: string;
    speak(): void;
}

class Dog implements Animal {
    constructor(public name: string) {}
    speak(): void {
        console.log(`${this.name} barks`);
    }
}

const myDog = new Dog("Buddy");
myDog.speak(); // Output: Buddy barks
```

Where Are Interfaces Used in TypeScript?

An interface in TypeScript is a way to define the structure of an object. It specifies what properties and methods an object should have but does not provide their implementation. Interfaces are used to ensure type safety, code consistency, and reusability in applications.

1. **Create a calculator in TypeScript that uses basic operations like addition, subtraction, multiplication, and division. It also gracefully handles invalid operations and division by zero..**

Code :-

```
class Calculator {  
    add(a: number, b: number): number {  
        return a + b;  
    }  
    subtract(a: number, b: number): number {  
        return a - b;  
    }  
    multiply(a: number, b: number): number {  
        return a * b;  
    }  
    divide(a: number, b: number): number | string {  
        if (b === 0) {  
            return "Error: Division by zero is not allowed";  
        }  
        return a / b;  
    }  
    calculate(a: number, b: number, operation: string): number | string {  
        switch (operation) {  
            case "add":  
                return this.add(a, b);  
            case "subtract":  
                return this.subtract(a, b);  
            case "multiply":  
                return this.multiply(a, b);  
            case "divide":  
                return this.divide(a, b);  
            default:  
                return "Error: Invalid operation";  
        }  
    }  
}
```

```
    }  
  }  
}  
  
const calc = new Calculator();  
console.log(calc.calculate(10, 5, "add"));  
console.log(calc.calculate(10, 5, "divide"));  
console.log(calc.calculate(10, 0, "divide"));  
console.log(calc.calculate(10, 5, "modulus"));
```

Output

```
[LOG]: 15  
[LOG]: 2  
[LOG]: "Error: Division by zero is not allowed"  
[LOG]: "Error: Invalid operation"
```

2. Design a Student Result database management system using TypeScript.

Code :-

```
class Student {  
  id: number;  
  name: string;  
  marks: { [subject: string]: number };  
  
  constructor(id: number, name: string) {  
    this.id = id;  
    this.name = name;  
  }  
}
```

```

    this.marks = {};
}

addMarks(subject: string, mark: number) {
    if (mark < 0 || mark > 100) {
        console.log(`Error: Marks for ${subject} should be between 0 and 100.`);
        return;
    }
    this.marks[subject] = mark;
}

getAverage(): number {
    const values = Object.values(this.marks);
    if (values.length === 0) return 0;
    return values.reduce((sum, mark) => sum + mark, 0) / values.length;
}

displayResult(): void {
    console.log(`\nStudent ID: ${this.id}`);
    console.log(`Name: ${this.name}`);
    console.log("Marks:", this.marks);
    console.log(`Average Marks: ${this.getAverage().toFixed(2)}`);
}
}

class StudentDatabase {
    students: Student[] = [];

    addStudent(id: number, name: string): void {
        if (this.students.some(student => student.id === id)) {
            console.log("Error: Student ID already exists.");
            return;
        }
        this.students.push(new Student(id, name));
    }
}

```

```

    }

    getStudent(id: number): Student | undefined {
        return this.students.find(student => student.id === id);
    }

    updateMarks(id: number, subject: string, mark: number): void {
        const student = this.getStudent(id);

        if (!student) {
            console.log("Error: Student not found.");
            return;
        }

        student.addMarks(subject, mark);
    }

    displayAllResults(): void {
        console.log("\n--- Student Results ---");
        this.students.forEach(student => student.displayResult());
    }
}

const db = new StudentDatabase();
db.addStudent(1, "Amit");
db.addStudent(2, "Priya");
db.updateMarks(1, "Math", 90);
db.updateMarks(1, "Science", 85);
db.updateMarks(2, "Math", 75);
db.updateMarks(2, "Science", 80);
db.displayAllResults();

```

output

```
[LOG]: "  
Student ID: 1"
```

```
[LOG]: "Name: Bhagyesh "
```

```
[LOG]: "Marks:", {  
  "Math": 90,  
  "Science": 85  
}
```

```
[LOG]: "Average Marks: 87.50"
```

```
[LOG]: "  
Student ID: 2"
```

```
[LOG]: "Name: Ram "
```

```
[LOG]: "Marks:", {  
  "Math": 75,  
  "Science": 80  
}
```

```
[LOG]: "Average Marks: 77.50"
```