Experiment – 1 a: TypeScript

Name of Student	**Atharva Prabhu**
Class Roll No	**42**
D.O.P.	**23/1/25**
D.O.S.	**30/1/25**
Sign and Grade	

Experiment – 1 a: TypeScript

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

Problem Statement:

a.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..

```
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(operation: string, a: number, b: number): 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. Supported operations are add, subtract, multiply, and
divide.";
        }
    }
}

const calc = new Calculator();
console.log(calc.calculate("add", 5, 3));
console.log(calc.calculate("subtract", 10, 4));
console.log(calc.calculate("multiply", 6, 7));
console.log(calc.calculate("divide", 20, 5));
console.log(calc.calculate("divide", 10, 0));
console.log(calc.calculate("modulus", 10, 3));
```

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

```
// Step 1: Declare basic data types
const studentName: string = "John Doe";
const subject1: number = 45;
const subject2: number = 38;
const subject3: number = 50;

// Step 2: Calculate the average marks
const totalMarks: number = subject1 + subject2 + subject3;
const averageMarks: number = totalMarks / 3;

// Step 3: Determine if the student has passed or failed
const isPassed: boolean = averageMarks >= 40;

// Step 4: Display the result
console.log(Student Name: ${studentName});
console.log(Average Marks: ${averageMarks});
console.log(Result: ${isPassed ? "Passed" : "Failed"});
```

Theory:

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

Ans: Data types in TypeScript are divided into 2 parts: Builtin and user-defined
Builtin

- String
- Number
- Boolean
- Void
- Symbol
- Null & Undefined

User-Defined

- Arrays
- Enums
- Classes
- Interfaces

b. How do you compile TypeScript files?

Ans: To compile TypeScript files, you need to use the TypeScript compiler (tsc). Follow these steps:

Install TypeScript globally if not installed:
sh
CopyEdit
npm install -g typescript

Compile a single TypeScript file:
sh
CopyEdit
tsc filename.ts

Compile multiple TypeScript files:
sh
CopyEdit
tsc file1.ts file2.ts

If you have a tsconfig.json file, compile all files in the project using:
sh
CopyEdit
tsc

This will use the configurations defined in tsconfig.json.

c. What is the difference between JavaScript and TypeScript?

Feature	JavaScript	TypeScript
Type System	Dynamically typed (no type checking)	Statically typed (types are enforced at compile time)
Compilation	Interpreted by the browser	Compiled to JavaScript using `tsc`
Error Checking	Errors occur at runtime	Errors are detected at compile time
Interfaces	Not supported	Supported (helps in defining structures)
OOP Features	Supports classes (ES6+) but limited	Fully supports OOP with interfaces, generics, access modifiers, etc.
Code Readability	More flexible but harder to maintain	More structured, making it easier to maintain

d. Compare how Javascript and Typescript implement Inheritance.

JavaScript Inheritance (Prototypal Inheritance)

JavaScript primarily uses prototype-based inheritance. Example:

```
function Animal(name) {
    this.name = name;
}
Animal.prototype.speak = function() {
    console.log(`${this.name} makes a noise.`);
};

function Dog(name, breed) {
    Animal.call(this, name);
    this.breed = breed;
}
Dog.prototype = Object.create(Animal.prototype);
Dog.prototype.constructor = Dog;

const dog = new Dog("Buddy", "Golden Retriever");
dog.speak();  // Buddy makes a noise.
```

TypeScript Inheritance (Class-based Inheritance)

TypeScript provides class-based inheritance with extends:

```
class Animal {
    name: string;
    constructor(name: string) {
        this.name = name;
    }
    speak(): void {
        console.log(`${this.name} makes a noise.`);
    }
}

class Dog extends Animal {
    breed: string;
    constructor(name: string, breed: string) {
        super(name);
        this.breed = breed;
    }
}

const dog = new Dog("Buddy", "Golden Retriever");
dog.speak(); // Buddy makes a noise.
```

e. How generics make the code flexible and why we should use generics over other types. In the lab assignment 3, why the usage of generics is more suitable than using any data type to handle the input.

Generics allow you to create reusable components that work with a variety of data types while maintaining type safety.

Feature	Generics (`T`)	`any`
Type Safety	Preserves type information	Loses type information
Code Readability	More readable and maintainable	Harder to debug and maintain
Performance	No unnecessary type conversions	May require type checking at runtime

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

Feature	Classes	Interfaces
Purpose	Defines how objects are created (blueprint)	Defines the structure of an object
Instantiation	Can be instantiated using `new`	Cannot be instantiated
Implementation	Can have methods with implementations	Only method signatures, no implementation
Access Modifiers	Supports (`public`, `private`, `protected`)	No access modifiers
Extensibility	Supports inheritance (`extends`)	Supports multiple interfaces (`implements`)

2. **Output:**

a.

```
PS C:\webx> node calculator.js
Atharva Prabhu D15A 42
8
6
42
4
Error: Division by zero is not allowed.
Error: Invalid operation. Supported operations are add, subtra
ct, multiply, and divide.
```

b.

```
PS C:\webx> node result.js
Atharva Prabhu D15A 42
Student Name: John Doe
Average Marks: 44.333333333333336
Result: Passed
```