Experiment – 1 a: TypeScript

Name of Student	**Rujuta Medhi**
Class Roll No	**D15A_27**
D.O.P.	**23/01/2025**
D.O.S.	**30/01/2025**
Sign and Grade	

- **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..

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?

- Primitive Types: number, string, boolean, null, undefined, bigint, symbol.
- Object Types: arrays, tuples, enums, interfaces, classes, functions.

- Special Types: any (disables type checking), unknown (safer alternative to any), void (for functions that return nothing), and never (for unreachable code).

Type annotations

Type annotations explicitly define variable types, making the code more predictable and reducing runtime errors.

Example:

```
let age: number = 25;
let isStudent: boolean = true;
let names: string[] = ["Alice", "Bob"];
```

b. How do you compile TypeScript files?

To convert TypeScript (.ts) files into JavaScript (.js), use the TypeScript compiler

```
tsc filename.ts
```

This generates a JavaScript file that can be executed in a browser or Node.js.

c. What is the difference between JavaScript and TypeScript?

Feature	JavaScript (JS)	TypeScript (TS)
Typing	Dynamic (no type checking)	Static (explicit types)
Compilation	No compilation needed	Compiles to JavaScript
Error Detection	Errors found at runtime	Errors caught during development
Object-Oriented Features	Uses prototypes	Supports classes, interfaces, generics
Code Maintainability	Harder to scale	Easier for large projects
ES6+ Support	Supports modern JS features	Includes ES6+ and ext ures
Browser Support	Directly supported	Needs compilation to JS

d. Compare how Javascript and Typescript implement Inheritance.

- JavaScript: Uses prototype-based inheritance. Objects inherit from other objects via prototypes.
- TypeScript: Uses class-based inheritance, similar to OOP languages like Java and C#. Supports access modifiers (public, private, protected).

Example in TypeScript:

```
class Animal {
  name: string;
  constructor(name: string) {
    this.name = name;
```

```
  }
}
class Dog extends Animal {
  bark() {
    console.log("Woof!");
  }
}
```

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 **write reusable and type-safe** code without sacrificing flexibility. They let you define **placeholder types** that can be used with different data types while maintaining strict type safety.

Key Benefits of Generics

Reusability → Write one function or class that works with multiple data types.
Type Safety → Prevents runtime type errors by ensuring type correctness at compile time.
Better Code Maintainability → Reduces code duplication and improves readability.

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

Feature	Class	Interface
Definition	Blueprint for creating objects with methods & properties	Defines a structure for objects without implementation
Usage	Used to create instances (objects)	Used for type-checking and enforcing structure
Implementation	Can have both properties & methods with implementations	Only has property and method signatures (no implementation)
Inheritance	Supports inheritance using `extends`	Supports multiple inheritance using `extends` or `implements`
Compiles to	JavaScript functions and constructors	No JS output, only used for type-checking
Accessibility Modifiers	Supports `public`, `private`, `protected`	Doesn't support access modifiers

Interfaces are mainly used for:

- **Defining Object Shapes** (e.g., API responses, function parameters)
- **Ensuring Class Structure** (via implements)

- **Enforcing Type Safety** in function arguments and return types

- **Output:**

A.

```
class Calculator {
   add(a: number, b: number): string {
      return `The addition of ${a} and ${b} is ${a + b}.`;
   }

   subtract(a: number, b: number): string {
      return `The subtraction of ${a} and ${b} is ${a - b}.`;
   }

   multiply(a: number, b: number): string {
       return `The multiplication of ${a} and ${b} is ${a * b}.`;
   }

   divide(a: number, b: number): number {
      if (b === 0) {
         throw new Error("Error: Division by zero is not allowed.");
      }
      return a / b;
   }

   calculate(operation: string, a: number, b: number): string | number {
      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:
            throw new Error("Error: Invalid operation.");
      }
   }
}

// Example Usage
```

```
const calc = new Calculator();

console.log(calc.calculate("add", 10, 5));  // Output: The addition of 10 and 5 is 15.
console.log(calc.calculate("subtract", 100, 5));  // Output: 95
console.log(calc.calculate("multiply", 10, 5));
console.log(calc.calculate("divide", 100, 0));
```

Output

```
The addition of 10 and 5 is 15.
The subtraction of 100 and 5 is 95.
The multiplication of 10 and 5 is 50.
/tmp/main.ts:16
            throw new Error("Error: Division by zero is not allowed.");
            ^


Error: Error: Division by zero is not allowed.
    at Calculator.divide (/tmp/main.ts:16:19)
    at Calculator.calculate (/tmp/main.ts:30:29)
```

B.

```
const studentName: string = "Rujuta Medhi";
const subject1: number = 85;
const subject2: number = 88;
const subject3: number = 90;

// 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"}`);
```

Output:

```
Student Name: Rujuta Medhi
Average Marks: 87.66666666666667
Result: Passed
```