Experiment – 1 a: TypeScript

Name of Student	**Pranav Pol**
Class Roll No	**D15A / 41**
D.O.P.	
D.O.S.	
Sign and Grade	

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

2. **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..

Code:

```
function calculator(a: number, b: number, operation: string): number | string {
    switch (operation) {
        case "add":
            return a + b;
        case "subtract":
            return a - b;
        case "multiply":
            return a * b;
        case "divide":
            return b !== 0 ? a / b : "Error: Division by zero is not allowed";
        default:
            return "Error: Invalid operation";
    }
}
console.log("code by pranav 41")
// Example usage:
console.log(calculator(10, 5, "add"));       // Output: 15
console.log(calculator(10, 5, "subtract"));  // Output: 5
```

```
console.log(calculator(10, 5, "multiply"));  // Output: 50
console.log(calculator(10, 0, "divide"));    // Output: Error: Division by zero is not allowed
console.log(calculator(10, 5, "modulus"));   // Output: Error: Invalid operation
```

Output:

```
code by pranav 41
15
5
50
Error: Division by zero is not allowed
Error: Invalid operation
```

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

```
// Step 1: Declare basic data types

const studentName: string = "Pranav Pol";

const subject1: number = 80;

const subject2: number = 95;

const subject3: number = 90;


// Step 2: Calculate the total and 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.toFixed(2)}`);

console.log(`Result: ${isPassed ? "Passed" : "Failed"}`);
```

Output:

```
Student Name: Pranav Pol
Average Marks: 88.33
Result: Passed
```

1. **Theory:**

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

Ans:

Different Data Types in TypeScript

TypeScript supports several data types, including:

- **Primitive Types:** number, string, boolean, bigint, symbol, null, undefined
- **User-defined Types:** interface, class, enum, type
- **Advanced Types:** any, unknown, never, tuple, union, intersection, void

2. Type Annotations in TypeScript

Type annotations allow you to specify the type of a variable explicitly, such as:

1. let num: number = 10;

2. let username: string = "Alice";

3. let isActive: boolean = true;

b. How do you compile TypeScript files?

Ans:

To compile a TypeScript file (.ts) into JavaScript (.js), use:

tsc filename.ts

This generates a filename.js file, which can be run in any JavaScript environment.

c. What is the difference between JavaScript and TypeScript?

Feature	TypeScript	JavaScript
Typing	Provides static typing	Dynamically typed
Tooling	Comes with IDEs and code editors	Limited built-in tooling
Syntax	Similar to JavaScript, with additional features	Standard JavaScript syntax
Compatibility	Backward compatible with JavaScript	Cannot run TypeScript in JavaScript files
Debugging	Stronger typing can help identify errors	May require more debugging and testing
Learning curve	Can take time to learn additional features	Standard JavaScript syntax is familiar

d. Compare how Javascript and Typescript implement Inheritance.

Ans:

Inheritance in JavaScript vs. TypeScript

- JavaScript uses **prototypal inheritance**, where objects inherit directly from other objects.
- TypeScript supports **class-based inheritance**, similar to Java, using extends.

Example in TypeScript:

Typescript:

```
class Person {
    name: string;
    constructor(name: string) {
```

```
        this.name = name;
    }
}
class Student extends Person {
    rollNo: number;
    constructor(name: string, rollNo: number) {
        super(name);
        this.rollNo = rollNo;
    }
}
```

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.

Ans:

- Generics make the code reusable and type-safe. Instead of using any, generics ensure that the function or class works with multiple types without losing type safety. Example:

Typescript:

```
function identity<T>(arg: T): T {
    return arg;
}
console.log(identity<number>(10));
console.log(identity<string>("Hello"));
```

Generics are preferred over any because they preserve type information.

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

Ans:

Difference Between Classes and Interfaces

- **Class**: A blueprint for creating objects, supports inheritance.
- **Interface**: Defines a structure but does not provide implementation.

Example:

typescript

```
interface Person {
    name: string;
    age: number;
}


class Student implements Person {
    name: string;
    age: number;
    constructor(name: string, age: number) {
        this.name = name;
        this.age = age;
    }}
```

Interfaces are used for type checking and defining the structure of an object without implementing it.

Conclusion:

This experiment successfully showcased the development of a **calculator** and a **student result management system** using TypeScript.

The **calculator** efficiently performs arithmetic operations while ensuring proper error handling, such as managing invalid inputs and preventing division by zero.

The **student result management system** effectively organizes student data, computes total and average marks, and determines pass/fail status using object-oriented programming principles.