Experiment – 1 a: TypeScript

Name of Student	**Mohit Patil**
Class Roll No	**36**
D.O.P.	**6/2/2025**
D.O.S.	
Sign and Grade	

Experiment – 1 a: TypeScript

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..

    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"});
```

3. **Theory:**

a. What are the different data types in TypeScript? What are Type Annotations in Typescript?
b. How do you compile TypeScript files?
c. What is the difference between JavaScript and TypeScript?
d. Compare how Javascript and Typescript implement Inheritance.
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.
f. What is the difference between Classes and Interfaces in Typescript? Where are interfaces used?

1. Different Data Types in TypeScript

Primitive Types: string, number, boolean, bigint, symbol, null, undefined

Object Types: object, array, tuple, enum, function

Special Types: any, unknown, never, void

User-defined Types: interface, class, type alias, union, intersection

2. Type Annotations in TypeScript

Used to explicitly define the data type of variables, function parameters, and return values.

Example:

```
let age: number = 25;

function greet(name: string): string {

  return "Hello, " + name;

}
```

3. How to Compile TypeScript Files?

- Use the TypeScript compiler (tsc) to compile .ts files into .js files.
  - Basic command:
    tsc filename.ts

- To watch changes continuously:
  tsc filename.ts --watch
- To compile multiple files:
  tsc

4. Difference Between JavaScript and TypeScript

Feature	JavaScript	TypeScript
Typing	Dynamically typed	Statically typed
Compilation	No compilation needed	Compiles to JavaScript
OOP Support	Limited	Strong OOP support
Error Handling	Errors runtime	Errors at compile-time
Code Maintainability	Less structured	More structured

5. Inheritance in JavaScript vs TypeScript

- JavaScript:
  - Uses prototype inheritance.
  - class was introduced in ES6, but it's syntactical sugar over prototypes.

Example:

```
class Animal {

 constructor(name) {

  this.name = name;

 }

 speak() {

  console.log(`${this.name} makes a sound`);
```

```
  }
}

class Dog extends Animal {
  speak() {
    console.log(`${this.name} barks`);
  }
}
```

- TypeScript:
  - Uses class with strong type checking.
  - Supports access modifiers (public, private, protected).

Example:

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

class Dog extends Animal {
  speak(): void {
    console.log(`${this.name} barks`);
  }
}
```

6. Generics in TypeScript

Make code more flexible and reusable by allowing types to be specified later.

Advantages over any:

Preserves type safety.

Avoids unnecessary type casting.

Makes code more maintainable.

Example of Generics:

```
function identity<T>(arg: T): T {
  return arg;
}
```

Why Use Generics in Lab Assignment 3 Instead of any?

Using any loses type information, leading to potential runtime errors.

Generics ensure that the input and output types remain consistent.

Example: Instead of

```
function processData(data: any): any { return data; }
```

Use

```
function processData<T>(data: T): T { return data; }
```

This allows processData to work correctly for different data types while maintaining type safety.

7. Difference Between Classes and Interfaces in TypeScript

Feature	Class	Interface
Definition	Blueprint for objects	Defines structure (not implementation)
Implementation	Can contain actual logic	No implementation, only structure
Inheritance	Can extend another class	Can extend multiple interfaces
Usage	Used to create objects	Used for type checking

- Where Are Interfaces Used?

Used for defining object shapes:

```
interface User {
```

```
  name: string;
  age: number;
}
let user: User = { name: "Alice", age: 25 };
```

Used for implementing contracts in classes:

```
interface Animal {
  speak(): void;
}
class Dog implements Animal {
  speak(): void {
    console.log("Bark!");
  }
}
```

- Used in function type definitions:

```
interface MathOperation {
  (x: number, y: number): number;
}
let add: MathOperation = (a, b) => a + b;
```

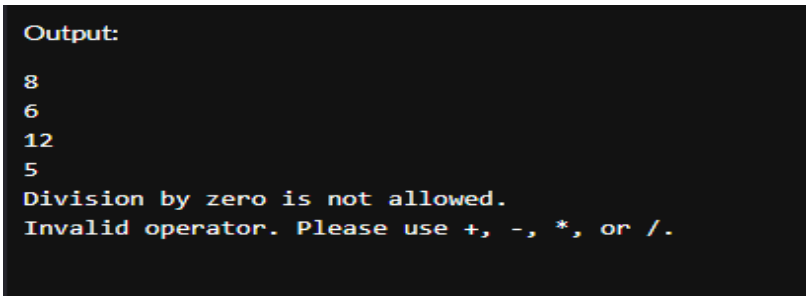
4. **Output:**

a.

```
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 {
    if (b === 0) {
      throw new Error("Division by zero is not allowed.");
    }
    return a / b;
  }
  calculate(operator: string, a: number, b: number): number | string {
    try {
      switch (operator) {
        case "+":
          return this.add(a, b);
        case "-":
          return this.subtract(a, b);
        case "*":
          return this.multiply(a, b);
        case "/":
          return this.divide(a, b);
        default:
          return "Invalid operator. Please use +, -, *, or /.";
      }
    } catch (error) {
      return error.message; // Return the error message for division by zero
    }
const calculator = new Calculator();
console.log(calculator.calculate("+", 5, 3));      // Output: 8
console.log(calculator.calculate("-", 10, 4));     // Output: 6
console.log(calculator.calculate("*", 2, 6));      // Output: 12
```

```
console.log(calculator.calculate("/", 10, 2));     // Output: 5
console.log(calculator.calculate("/", 10, 0));     // Output: Division by zero is not allowed.
console.log(calculator.calculate("%", 5, 2));      // Output: Invalid operator. Please use +, -, *, or /.
```



b.

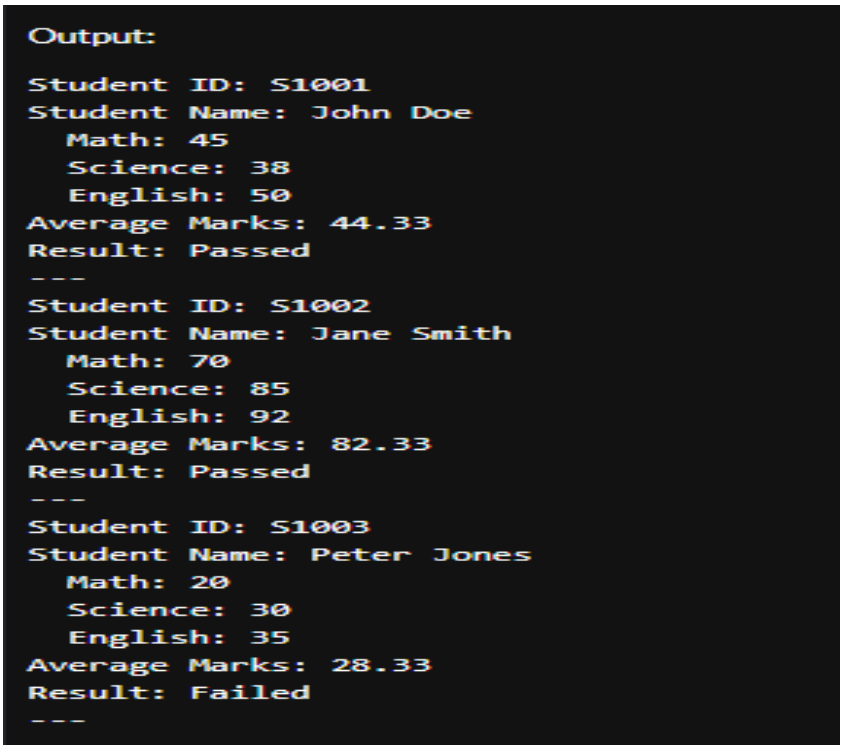
code:

```
interface Subject {
  name: string;
  marks: number;
}
interface Student {
  id: string;
  name: string;
  subjects: Subject[];
}
interface Result {
  studentId: string;
  averageMarks: number;
  isPassed: boolean;
}
function calculateAverage(subjects: Subject[]): number {
  if (subjects.length === 0) {
    return 0;
  }
```

```
  const totalMarks = subjects.reduce((sum, subject) => sum + subject.marks, 0);
  return totalMarks / subjects.length;
}
function determinePassStatus(averageMarks: number, passingThreshold: number = 40): boolean {
  return averageMarks >= passingThreshold;
}
function generateResult(student: Student): Result {
  const averageMarks = calculateAverage(student.subjects);
  const isPassed = determinePassStatus(averageMarks);
  return {
    studentId: student.id,
    averageMarks: averageMarks,
    isPassed: isPassed,
  };
}
function displayResult(student: Student, result: Result): void {
  console.log(`Student ID: ${student.id}`);
  console.log(`Student Name: ${student.name}`);
  student.subjects.forEach(subject => {
    console.log(`  ${subject.name}: ${subject.marks}`);
  });
  console.log(`Average Marks: ${result.averageMarks.toFixed(2)}`);
  console.log(`Result: ${result.isPassed ? "Passed" : "Failed"}`);
  console.log("---");
}
const student1: Student = {
  id: "S1001",
  name: "John Doe",
  subjects: [
```

```
    { name: "Math", marks: 45 },
    { name: "Science", marks: 38 },
    { name: "English", marks: 50 },
  ],
};
const student2: Student = {
  subjects: [
    { name: "Math", marks: 70 },
    { name: "Science", marks: 85 },
    { name: "English", marks: 92 },
  ],
};
const result1 = generateResult(student1);
const result2 = generateResult(student2);
displayResult(student1, result1);
displayResult(student2, result2);
const student3: Student = {
    id: "S1003",
    name: "Peter Jones",
    subjects: [
      {name: "Math", marks: 20},
      {name: "Science", marks: 30},
      {name: "English", marks: 35}
    ]
  };
  const result3 = generateResult(student3);
  displayResult(student3, result3);
```

output:


Output:
Student ID: S1001
Student Name: John Doe
Math: 45
Science: 38
English: 50
Average Marks: 44.33
Result: Passed

Student ID: S1002
Student Name: Jane Smith
Math: 70
Science: 85
English: 92
Average Marks: 82.33
Result: Passed

Student ID: S1003
Student Name: Peter Jones
Math: 20
Science: 30
English: 35
Average Marks: 28.33
Result: Failed
