

**Experiment – 1 a: TypeScript**

<b>Name of Student</b>	<b><u>Tejas Dhondibhau Gunjal</u></b>
<b>Class Roll No</b>	<b><u>18</u></b>
<b>D.O.P.</b>	<b><u>28-01-2025</u></b>
<b>D.O.S.</b>	<b><u>11-02-2025</u></b>
<b>Sign and Grade</b>	

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?

Data Types in TypeScript:

TypeScript offers a rich set of data types, including:

- Primitive types like string, number, boolean, null, undefined, symbol, and bigint.
- Special types like any (for any value), unknown (safer than any), void (for functions that don't return a value), and never (for values that never occur, such as errors).
- Complex types like object, arrays, and tuple (a fixed-size collection of elements with different types).

**Type Annotations:** Type annotations allow developers to specify types for variables, function parameters, and return values. This enables TypeScript's static type checking, improving code safety and readability. Example: let age: number = 30;

- b. How do you compile TypeScript files?

1. Install TypeScript via npm: npm install -g typescript.
2. Compile: Use tsc filename.ts to compile TypeScript to JavaScript.
3. Watch mode: Run tsc --watch for automatic compilation upon changes.
4. Using tsconfig.json: Create a configuration file to manage compilation settings.

- c. What is the difference between JavaScript and TypeScript?

Feature	JavaScript	TypeScript
<b>Typing</b>	Dynamically typed (no static types)	Statically typed with optional type annotations
<b>Compilation</b>	Interpreted directly by the browser	Needs to be compiled to JavaScript using tsc
<b>Type Checking</b>	No type checking at compile time	Type checking happens at compile time
<b>Support for ES Features</b>	Supports ES5 features (older versions)	Supports ES6/ES7 and beyond (Including features like async/await, decorators, etc.)

<b>Error Handling</b>	Errors are only found at runtime	Errors can be caught at compile time
<b>Object-Oriented Support</b>	Limited (using prototypes)	Full support (classes, interfaces, inheritance)
<b>Interoperability</b>	Runs directly in the browser or Node	Needs to be compiled to JavaScript first

- d. Compare how Javascript and Typescript implement Inheritance.

**JavaScript** uses prototype-based inheritance, where objects can inherit properties and methods from other objects.

**TypeScript** supports class-based inheritance and offers features like access modifiers (public, private), and type safety. It can also implement interfaces for stricter contracts between classes.

- 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 writing functions or classes that can work with any data type while maintaining type safety. This makes code flexible and reusable, as it can handle multiple types without sacrificing type safety. Generics avoid the use of any, which bypasses type checking and could lead to runtime errors.

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

Classes are blueprints for creating objects with properties and methods. They can have constructors, implement interfaces, and inherit from other classes.

Interfaces define the structure of an object, specifying which properties and methods must exist, without providing any implementation. Interfaces are used to ensure that classes or objects adhere to a certain contract. Interfaces are typically used for defining the shape of objects, for function signatures, or for establishing contracts in class-based architecture. They ensure type consistency across various parts of the application.

#### 4. GitHub Link: - [https://github.com/tejasgunjal021/WEBX\\_1A](https://github.com/tejasgunjal021/WEBX_1A)

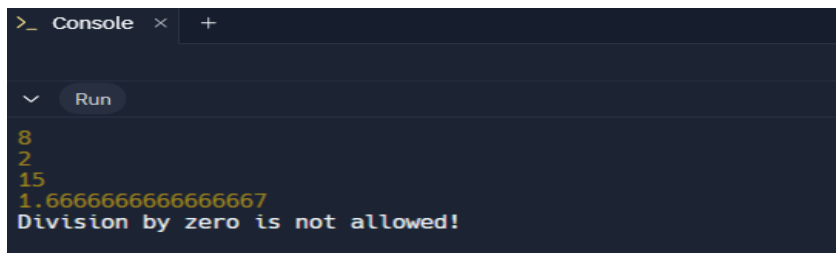
## 5. Output:

### PS A)

```
function calculator(a: number, b: number, operator: string): number | never {
  switch (operator) {
    case "+":
      return a + b;
    case "-":
      return a - b;
    case "*":
      return a * b;
    case "/":
      if (b === 0) {
        throw new Error("Division by zero is not allowed!");
      }
      return a / b;
    default:
      throw new Error(`Invalid operator: '${operator}'. Use +, -, *, or /.`);
  }
}

try {
  console.log(calculator(5, 3, "+"));
  console.log(calculator(5, 3, "-"));
  console.log(calculator(5, 3, "*"));
  console.log(calculator(5, 3, "/"));
  console.log(calculator(7, 0, "/"));
  console.log(calculator(10, 2, "%"));
} catch (error) {
  console.error((error as Error).message);
}
```

### Output: -



```
>_ Console x +
Run
8
2
15
1.6666666666666667
Division by zero is not allowed!
```

**PS B)**

```

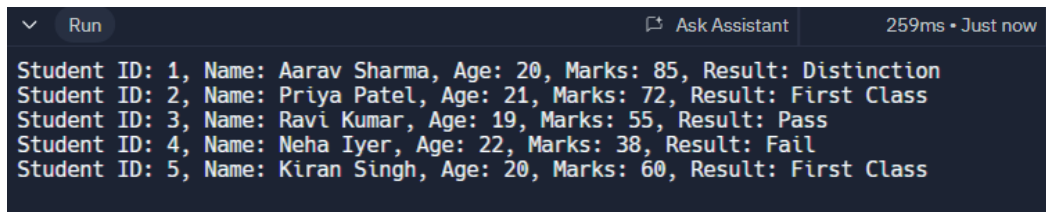
type Student = {
  id: number;
  name: string;
  surname: string;
  age: number;
  marks: number;
};

const students: Student[] = [
  { id: 1, name: "Aarav", surname: "Sharma", age: 20, marks: 85 },
  { id: 2, name: "Priya", surname: "Patel", age: 21, marks: 72 },
  { id: 3, name: "Ravi", surname: "Kumar", age: 19, marks: 55 },
  { id: 4, name: "Neha", surname: "Iyer", age: 22, marks: 38 },
  { id: 5, name: "Kiran", surname: "Singh", age: 20, marks: 60 },
];

const determineResult = (marks: number): string => {
  if (marks < 40) return "Fail";
  if (marks < 60) return "Pass";
  if (marks < 75) return "First Class";
  return "Distinction";
};

students.forEach((student) => {
  const result = determineResult(student.marks);
  console.log(
    `Student ID: ${student.id}, Name: ${student.name} ${student.surname}, Age: ${student.age}, Marks:
    ${student.marks}, Result: ${result}`,
  );
});

```

**Output: -**


```

Student ID: 1, Name: Aarav Sharma, Age: 20, Marks: 85, Result: Distinction
Student ID: 2, Name: Priya Patel, Age: 21, Marks: 72, Result: First Class
Student ID: 3, Name: Ravi Kumar, Age: 19, Marks: 55, Result: Pass
Student ID: 4, Name: Neha Iyer, Age: 22, Marks: 38, Result: Fail
Student ID: 5, Name: Kiran Singh, Age: 20, Marks: 60, Result: First Class

```

**Conclusion : -**

The TypeScript programs demonstrate the use of basic data types (number, string, boolean) and operators in practical applications. The calculator efficiently performs arithmetic operations while handling errors like division by zero. The Student Result database system processes student marks, calculates the average, and determines the pass/fail status based on a threshold. These implementations highlight TypeScript's type safety and logical operations, ensuring reliability and clarity in programming.