

### EXPERIMENT– 1 a: TypeScript

Name of Student	Sonam Chhabaidiya
Class Roll No	D15A_09
D.O.P.	
D.O.S.	
Sign and Grade	

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

**Problem Statement:**

- 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..
- 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

### 1. What are the different data types in TypeScript?

TypeScript provides several built-in data types:

- **Primitive Types:** `string`, `number`, `boolean`, `null`, `undefined`, `symbol`, `bigint`
- **Object Types:** `Array`, `Tuple`, `Enum`, `Class`, `Interface`
- **Special Types:** `any`, `unknown`, `void`, `never`

### 2. What are Type Annotations in TypeScript?

Type annotations allow developers to explicitly define variable types, function parameters, or return values.

Example:

```
let age: number = 25;    // Ensures 'age' holds only numeric values
```

### 3. How do you compile TypeScript files?

Use the TypeScript compiler (`tsc`) to convert TypeScript files into JavaScript:

```
tsc filename.ts
```

This generates a `filename.js` file for execution in a browser or Node.js.

### 4. Difference between JavaScript and TypeScript

Feature	JavaScript	TypeScript
Typing	Dynamic	Static (compile-time checking)
Interfaces	Not supported	Supported for structured code
Compilation	Interpreted	Compiled to JavaScript
Inheritance	Prototype-based	Class-based (OOP style)

### 5. JavaScript vs. TypeScript in Inheritance

- **JavaScript:** Uses prototype-based inheritance.
- **TypeScript:** Uses class-based inheritance (`class` and `extends` keywords), making it structured and maintainable.

## 6. Why Use Generics in TypeScript?

Generics provide **type safety** and **flexibility** while preventing runtime errors.

Example:

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

Using `any` instead of generics makes the code error-prone.

## 7. Classes vs. Interfaces in TypeScript

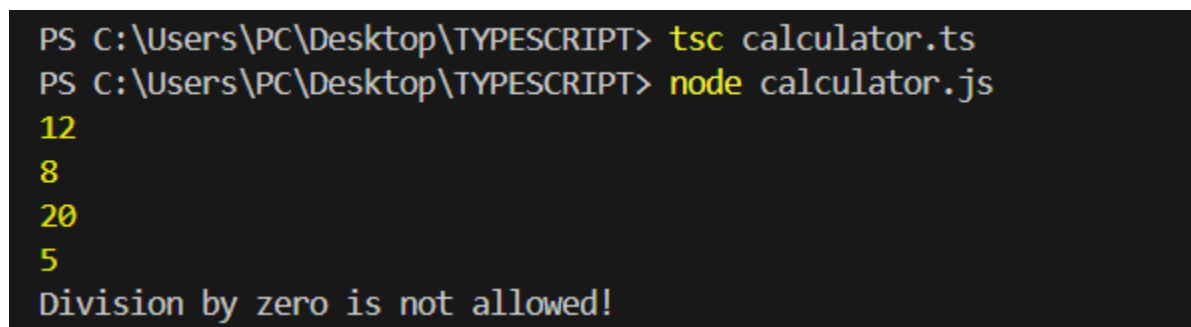
Feature	Class	Interface
Definition	Defines properties and methods	Defines structure but no implementation
Instantiation	Can create objects	Cannot be instantiated
Use Case	Used to implement functionality	Used to define object contracts

GitHub Link: <https://github.com/sonamcc/webx1a>

### Output

#### (a) TypeScript Calculator

Output:



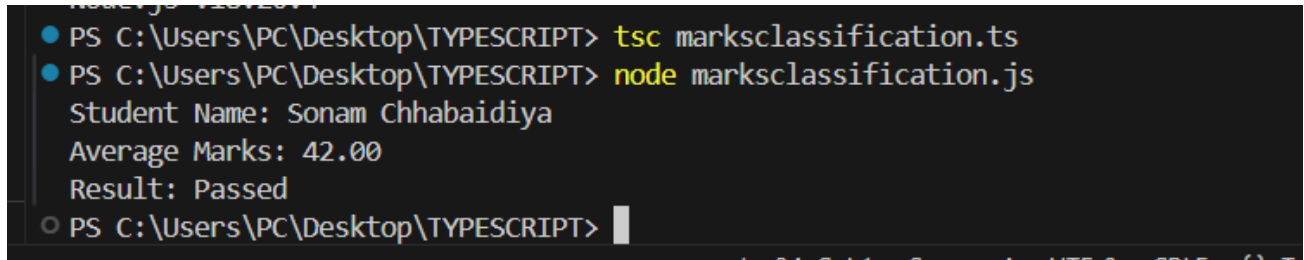
```
PS C:\Users\PC\Desktop\TYPESCRIPT> tsc calculator.ts  
PS C:\Users\PC\Desktop\TYPESCRIPT> node calculator.js  
12  
8  
20  
5  
Division by zero is not allowed!
```

This screenshot showcases the output of the **TypeScript Calculator**, which performs basic arithmetic operations such as addition, subtraction, multiplication, and division. The console displays:

- The results of valid operations (`add`, `subtract`, `multiply`, and `divide`).
- An error message when attempting **division by zero**.
- An error message for an **invalid operation** (e.g., `modulus`).

## (b) Student Result Database Management System

### Output:



```

PS C:\Users\PC\Desktop\TYPESCRIPT> tsc marksclassification.ts
PS C:\Users\PC\Desktop\TYPESCRIPT> node marksclassification.js
Student Name: Sonam Chhabaidiya
Average Marks: 42.00
Result: Passed
PS C:\Users\PC\Desktop\TYPESCRIPT>

```

This screenshot displays the output of the **Student Result Database Management System**, which calculates and prints:

- The **student's name**.
- The **average marks** (formatted to two decimal places).
- The **final result** (either "Passed" or "Failed" based on a 40% passing criteria).

## 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.