

Experiment – 1 a: TypeScript

Name of Student	<u>Shreyash Ganesh Dhekane</u>
Class Roll No	<u>16</u>
D.O.P.	<u>28-01-2025</u>
D.O.S.	<u>11-02-2025</u>
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:

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

TypeScript has basic types like `string`, `number`, `boolean`, `null`, `undefined`, `void`, `any`, `unknown`, `never`, `object`, arrays (`number[]` or `Array<number>`), tuples (`[string, number]`), and enums.

Type annotations explicitly define the type of a variable, function parameter, or return value.

Example:

```
let age: number = 25;
function greet(name: string): string {
    return "Hello, " + name;
}
```

2. How do you compile TypeScript files?

Use the TypeScript compiler (tsc):

```
tsc filename.ts
```

This generates a JavaScript file (filename.js) from the TypeScript code.

3. What is the difference between JavaScript and TypeScript?

- JavaScript is dynamically typed; TypeScript is statically typed.
- TypeScript has optional type annotations, interfaces, and generics.
- TypeScript requires compilation to JavaScript before execution.
- TypeScript provides better tooling and error checking at compile time.

4. Compare how Javascript and Typescript implement Inheritance.

- JavaScript uses prototype-based inheritance:

```
function Person(name) {
    this.name = name;
}
Person.prototype.greet = function () {
    console.log("Hello " + this.name);
};
```

- TypeScript uses class-based inheritance:

```
class Person {
    constructor(public name: string) {}
    greet() {
        console.log("Hello " + this.name);
    }
}
class Student extends Person {
    constructor(name: string, public grade: number) {
        super(name);
    }
}
```

5. 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 reusable and type-safe functions/classes.
- They ensure the correct type is maintained across different uses.

Example:

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

- Why generics are better?
 - any removes type safety, allowing unintended values.
 - Generics retain type information, ensuring better reliability.
6. What is the difference between Classes and Interfaces in Typescript? Where are interfaces used?
- Classes define implementation and behavior; they can have constructors and methods.
 - Interfaces define a structure (contract) without implementation.
 - Where are interfaces used?
 - Used for defining object shapes, function types, and enforcing structure in TypeScript.

Example:

```
interface User {
    name: string;
    age: number;
}
function getUser(user: User) {
    console.log(user.name);
}
```

Output:

1. 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..

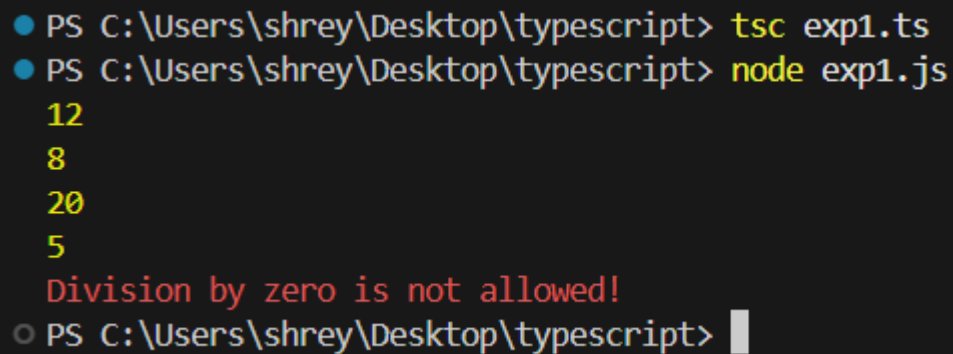
```
function calculator(a: number, b: number, operator: string): number | never {
    switch (operator) {
        case "+":
            return a + b;
        case "-":
            return a - b;
        case "*":
            return a * b;
        case "/":
            if (b === 0) {
                throw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

// Example Usage

```

try {
    console.log(calculator(10, 2, "+")); // Output: 12
    console.log(calculator(10, 2, "-")); // Output: 8
    console.log(calculator(10, 2, "*")); // Output: 20
    console.log(calculator(10, 2, "/")); // Output: 5
    console.log(calculator(10, 0, "/")); // Throws Error: Division by zero is not allowed!
    console.log(calculator(10, 2, "%")); // Throws Error: Invalid operator
} catch (error) {
    console.error((error as Error).message);
}

```



```

● PS C:\Users\shrey\Desktop\typescript> tsc exp1.ts
● PS C:\Users\shrey\Desktop\typescript> node exp1.js
12
8
20
5
Division by zero is not allowed!
○ PS C:\Users\shrey\Desktop\typescript>

```

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

```

// Define the type for a Student
type Student = {
    id: number;
    name: string;
    surname: string;
    age: number;
    marks: number;
};

// Array of student data with surnames
const students: Student[] = [
    { id: 1, name: "Shreyash", surname: "Dhekane", age: 20, marks: 85 },
    { id: 2, name: "Bob", surname: "Smith", age: 21, marks: 72 },
    { id: 3, name: "Charlie", surname: "Brown", age: 19, marks: 55 },
    { id: 4, name: "Diana", surname: "Williams", age: 22, marks: 38 },
    { id: 5, name: "Eve", surname: "Davis", age: 20, marks: 60 },
];

// Function to determine the result category of a student
const determineResult = (marks: number): string => {
    if (marks < 40) return "Fail";
    if (marks < 60) return "Pass";
    if (marks < 75) return "First Class";
    return "Distinction";
};

```

```
// Iterate through the students and display their result
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}`
    );
});
```

```
● PS C:\Users\shrey\Desktop\typescript> tsc exp2.ts
● PS C:\Users\shrey\Desktop\typescript> node exp2.js
Student ID: 1, Name: Shreyash Dhekane, Age: 20, Marks: 85, Result: Distinction
Student ID: 2, Name: Bob Smith, Age: 21, Marks: 72, Result: First Class
Student ID: 3, Name: Charlie Brown, Age: 19, Marks: 55, Result: Pass
Student ID: 4, Name: Diana Williams, Age: 22, Marks: 38, Result: Fail
Student ID: 5, Name: Eve Davis, Age: 20, Marks: 60, Result: First Class
○ PS C:\Users\shrey\Desktop\typescript> █
```