## Experiment – 1 a: TypeScript

| Name of Student | Jai Navani |
|---|---|
| Class Roll No | 30 |
| D.O.P. | 23/1/25 |
| D.O.S. | 30/1/25 |
| Sign and Grade | |

## Experiment – 1 a: TypeScript

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

**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..

```typescript
import * as promptSync from 'prompt-sync';

// Initializing prompt
const prompt = promptSync();

// Getting inputs from the user
const input1 = prompt("Enter first number: ");
const input2 = prompt("Enter second number: ");

// Converting inputs to numbers
const num1 = parseFloat(input1);
const num2 = parseFloat(input2);

// Performing calculations
console.log("Addition:", num1 + num2);
console.log("Subtraction:", num1 - num2);
console.log("Multiplication:", num1 * num2);
console.log("Division:", num1 / num2);
```

PROBLEMS 1    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

```
Found 1 error in calculator.ts:5

PS C:\Users\navan\Desktop\typescript-calculator> tsc calculator.ts
>>
PS C:\Users\navan\Desktop\typescript-calculator> node calculator.js
>>
Enter first number: 4
Enter second number: 5
Addition: 9
Subtraction: -1
Multiplication: 20
Division: 0.8
PS C:\Users\navan\Desktop\typescript-calculator>
```

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

```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"});
```

**OUTPUT:**

Student Name: John Doe

Average Marks: 44.333333333333336

Result: Passed

**Theory:**

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

Ans: Data types in TypeScript are divided into 2 parts: Builtin and user-defined
Builtin
- String
- Number
- Boolean
- Void
- Symbol
- Null & Undefined

User-Defined
- Arrays
- Enums
- Classes
- Interfaces

b.  How do you compile TypeScript files?

Ans: To compile TypeScript files, you need to use the TypeScript compiler (tsc). Follow these steps:

Install TypeScript globally if not installed:
sh
CopyEdit
npm install -g typescript

Compile a single TypeScript file:
sh
CopyEdit
tsc filename.ts

Compile multiple TypeScript files:
sh
CopyEdit
tsc file1.ts file2.ts

If you have a tsconfig.json file, compile all files in the project using:
sh
CopyEdit
tsc

This will use the configurations defined in tsconfig.json.

c. What is the difference between JavaScript and TypeScript?

| Feature | JavaScript | TypeScript |
|---|---|---|
| Type System | Dynamically typed (no type checking) | Statically typed (types are enforced at compile time) |
| Compilation | Interpreted by the browser | Compiled to JavaScript using `tsc` |
| Error Checking | Errors occur at runtime | Errors are detected at compile time |
| Interfaces | Not supported | Supported (helps in defining structures) |
| OOP Features | Supports classes (ES6+) but limited | Fully supports OOP with interfaces, generics, access modifiers, etc. |
| Code Readability | More flexible but harder to maintain | More structured, making it easier to maintain |

d. Compare how Javascript and Typescript implement Inheritance.

JavaScript Inheritance (Prototypal Inheritance)
JavaScript primarily uses prototype-based inheritance. Example:

```
function Animal(name) {
    this.name = name;
}
Animal.prototype.speak = function() {
    console.log(`${this.name} makes a noise.`);
};

function Dog(name, breed) {
    Animal.call(this, name);
    this.breed = breed;
}
Dog.prototype = Object.create(Animal.prototype);
Dog.prototype.constructor = Dog;

const dog = new Dog("Buddy", "Golden Retriever");
dog.speak();  // Buddy makes a noise.
```

TypeScript Inheritance (Class-based Inheritance)
TypeScript provides class-based inheritance with extends:

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

class Dog extends Animal {
    breed: string;
    constructor(name: string, breed: string) {
        super(name);
        this.breed = breed;
    }
}

const dog = new Dog("Buddy", "Golden Retriever");
dog.speak(); // Buddy makes a noise.
```

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 you to create reusable components that work with a variety of data types while maintaining type safety.

| Feature | Generics ( T ) | any |
| --- | --- | --- |
| Type Safety | Preserves type information | Loses type information |
| Code Readability | More readable and maintainable | Harder to debug and maintain |
| Performance | No unnecessary type conversions | May require type checking at runtime |

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

| Feature | Classes | Interfaces |
|---|---|---|
| Purpose | Defines how objects are created (blueprint) | Defines the structure of an object |
| Instantiation | Can be instantiated using `new` | Cannot be instantiated |
| Implementation | Can have methods with implementations | Only method signatures, no implementation |
| Access Modifiers | Supports (`public`, `private`, `protected`) | No access modifiers |
| Extensibility | Supports inheritance (`extends`) | Supports multiple interfaces (`implements`) |