

### Experiment – 1 a: TypeScript

Name of Student	Arnav Santosh Sawant
Class Roll No	D15A - 52
D.O.P.	06/02/2025
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.
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?

4. **Output:**

**<Include Code and Screenshot of Output>**

**Q1) Code and Output Screenshot**

*// Declaring basic data types*

let age: *number* = 25;

let vesit\_class: *string* = "D15A";

let isStudent: *boolean* = true;

*// Performing some operations*

let nextYearAge: *number* = age + 1;

let greeting: *string* = `Hello, i am from class \${vesit\_class} and I am \${age} years old.`;

*// Displaying output*

console.log(greeting);

console.log(`Next year, I will be \${nextYearAge} years old.`);

console.log(`Am I a student? \${isStudent}`);

**OUTPUT:-**

```
● PS C:\Users\ARNAV SAWANT\Desktop\WebX lab> cd .\exp1a\  
● PS C:\Users\ARNAV SAWANT\Desktop\WebX lab\exp1a> npx tsc question1.ts  
● PS C:\Users\ARNAV SAWANT\Desktop\WebX lab\exp1a> node question1.js  
Hello, i am from class D15A and I am 25 years old.  
Next year, I will be 26 years old.  
Am I a student? true
```

## Q2) Code and Output Screenshot

*// Calculator function*

```
function calculate(a: number, b: number, operation: string): number | string {
```

```
  switch (operation) {
```

```
    case "add":
```

```
      return a + b;
```

```
    case "subtract":
```

```
      return a - b;
```

```
    case "multiply":
```

```
      return a * b;
```

```
    case "divide":
```

```
      return b !== 0 ? a / b : "Error: Division by zero is not allowed";
```

```
    default:
```

```
      return "Error: Invalid operation";
```

```
  }
```

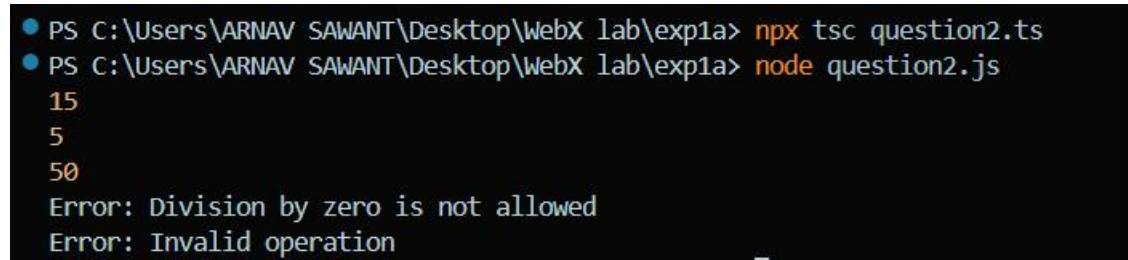
```
}
```

*// Test cases*

```
console.log(calculate(10, 5, "add"));    // 15
```

```
console.log(calculate(10, 5, "subtract")); // 5
console.log(calculate(10, 5, "multiply")); // 50
console.log(calculate(10, 0, "divide")); // Error: Division by zero is not allowed
console.log(calculate(10, 5, "modulus")); // Error: Invalid operation
```

### OUTPUT:-



```
PS C:\Users\ARNAV SAWANT\Desktop\WebX lab\exp1a> npx tsc question2.ts
PS C:\Users\ARNAV SAWANT\Desktop\WebX lab\exp1a> node question2.js
15
5
50
Error: Division by zero is not allowed
Error: Invalid operation
```

### Q3) Code and Output Screenshot

*// 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;
```

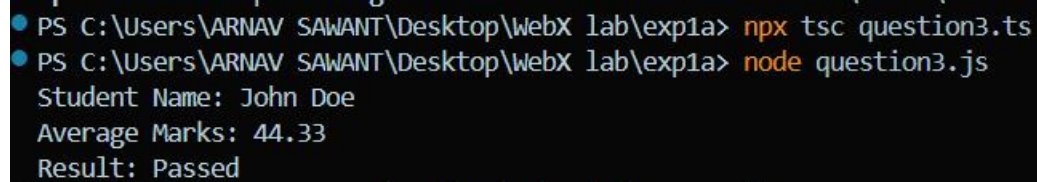
*// Step 3: Determine if the student has passed or failed*

```
const isPassed: boolean = averageMarks >= 40;
```

*// Step 4: Display the result*

```
console.log(`Student Name: ${studentName}`);  
  
console.log(`Average Marks: ${averageMarks.toFixed(2)}`);  
  
console.log(`Result: ${isPassed ? "Passed" : "Failed"}`);
```

### OUTPUT:-



```
PS C:\Users\ARNAV SAWANT\Desktop\WebX lab\exp1a> npx tsc question3.ts  
PS C:\Users\ARNAV SAWANT\Desktop\WebX lab\exp1a> node question3.js  
Student Name: John Doe  
Average Marks: 44.33  
Result: Passed
```

### THEORY ANSWERS :-

Ans Q1)

**Data types -**

#### Primitive Types

- number: let age: number = 25;
- string: let name: string = "D15A";
- boolean: let isStudent: boolean = true;
- null: let emptyValue: null = null;
- undefined: let notDefined: undefined = undefined;

#### Special Types

- any: let randomValue: any = "Hello";
- unknown: let someValue: unknown = "World";
- void (for functions without return):

```
function logMessage(): void {  
    console.log("No return value");  
}
```

### **Complex Types**

- array: `let numbers: number[] = [1, 2, 3];`
- tuple: `let person: [string, number] = ["John", 25];`
- enum:

```
enum Status { Success, Failure, Pending }
```

```
let currentStatus: Status = Status.Success;
```

- object:

```
let student: { name: string; age: number } = { name: "Alice", age: 21 };
```

### **Type Annotations:-**

Type annotations ensure type safety.

#### 1. Variable Annotation

```
let username: string = "Arnav";
```

```
let count: number = 10;
```

#### 2. Function Parameter & Return Type

```
function add(a: number, b: number): number {  
    return a + b;  
}
```

#### 3. Array Annotation

```
let scores: number[] = [85, 90, 78];
```

#### 4. Object Annotation

```
let person: { name: string; age: number } = { name: "John", age: 25 };
```

#### 5. Void Function

```
function greet(name: string): void {  
    console.log(`Hello, ${name}`);  
}
```

**Ans Q2)** To compile typescript files we have to follow the following steps:-

Step-1:- Install typescript globally in your system

```
npm i -g typescript
```

Step-2:- Once you write the code for a typescript file, then you compile it by writing..

```
npx tsc filename.ts
```

Step-3:- A filename.js file would be generated just simply run that file

```
node filename.js
```

**Ans Q3)**

Feature	JavaScript	TypeScript
Type Safety	No static typing	Supports static typing
Compilation	No compilation needed	Needs compilation ( <code>tsc</code> )
OOP Features	Limited (prototypes)	Supports classes, interfaces, generics
Readability	Can have runtime errors	Catches errors at compile time
Performance	Interpreted directly by browsers	Needs transpilation to JS

### Ans Q4)

Both JavaScript and TypeScript support **class-based inheritance**, but TypeScript enforces **strong type checking**.

### JavaScript Inheritance

Uses class with extends:

```
class Parent {  
    constructor(name) {  
        this.name = name;  
    }  
    greet() {  
        console.log(`Hello, ${this.name}`);  
    }  
}  
  
class Child extends Parent {  
    constructor(name, age) {  
        super(name);  
        this.age = age;  
    }  
}  
  
let obj = new Child("John", 21);  
  
obj.greet(); // Hello, John
```

### TypeScript Inheritance

Same syntax but with **type safety**:



```
class Parent {  
    name: string;  
    constructor(name: string) {  
        this.name = name;  
    }  
    greet(): void {  
        console.log(`Hello, ${this.name}`);  
    }  
}  
  
class Child extends Parent {  
    age: number;  
    constructor(name: string, age: number) {  
        super(name);  
        this.age = age;  
    }  
}  
  
let obj = new Child("John", 21);  
  
obj.greet(); // Hello, John
```

**Key Difference:**

- JavaScript allows any type of values.
- TypeScript enforces strict type checking.

**Ans Q5)**

## How Generics Improve Code Flexibility

Generics allow functions and classes to work with **multiple data types** while maintaining **type safety**.

Example with Generics:

```
function identity<T>(value: T): T {  
    return value;  
}  
  
console.log(identity<number>(10)); // 10  
  
console.log(identity<string>("Hello")); // Hello
```

Feature	Type	Generics
Type safety	No	Yes
Code reusability	Limited	High
Performance	Slightly lower	Optimized
Predictability	Can cause unexpected errors	Ensures correct data type

## Ans Q6)

Feature	Classes	Interfaces
Purpose	Blueprint for objects	Defines object structure
Implementation	Can have methods and properties	Only defines structure
Usage	Used to create instances	Used for type checking
Inheritance	Supports inheritance	Supports multiple inheritances

Where Are Interfaces Used?

- Defining object shapes

```
interface Person {  
    name: string;  
    age: number;  
}  
  
let user: Person = { name: "Arnav", age: 22 };
```

- Enforcing structure in classes

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

- Ensuring API response types

```
interface ApiResponse {  
    data: string;  
    status: number;  
}
```

Conclusion:

- Use classes when you need to create instances.
- Use interfaces when defining expected structures.