Name: Kaushik Kotian Roll no.:30 Div : D15B Batch : B

EXPERIMENT NO.:2

Theory:

JavaScript was introduced as a language for the client side. The development of Node.js has marked JavaScript as an emerging server-side technology too. However, as JavaScript code grows, it tends to get messier, making it difficult to maintain and reuse the code. Moreover, its failure to embrace the features of Object Orientation, strong type checking and compile-time error checks prevents JavaScript from succeeding at the enterprise level as a full-fledged server-side technology. TypeScript was presented to bridge this gap. TypeScript is both a language and a set of tools. TypeScript is a typed superset of JavaScript compiled to JavaScript. In other words, TypeScript is JavaScript plus some additional features.

Features of TypeScript

TypeScript is just JavaScript. TypeScript starts with JavaScript and ends with JavaScript. Typescript adopts the basic building blocks of your program from JavaScript. Hence, you only need to know JavaScript to use TypeScript. All TypeScript code is converted into its JavaScript equivalent for the purpose of execution.

TypeScript supports other JS libraries. Compiled TypeScript can be consumed from any JavaScript code. TypeScript-generated JavaScript can reuse all of the existing JavaScript frameworks, tools, and libraries.

JavaScript is TypeScript. This means that any valid **.js** file can be renamed to **.ts** and compiled with other TypeScript files.

TypeScript is portable. TypeScript is portable across browsers, devices, and operating systems. It can run on any environment that JavaScript runs on. Unlike its counterparts, TypeScript doesn't need a dedicated VM or a specific runtime environment to execute.

The benefits of TypeScript include –

- **Compilation** – JavaScript is an interpreted language. Hence, it needs to be run to test that it is valid. It means you write all the codes just to find no output, in case there is an error. Hence, you have to spend hours trying to find bugs in the code. The TypeScript transpiler provides the error-checking feature. TypeScript will compile the code and generate compilation errors, if it finds some sort of syntax errors. This helps to highlight errors before the script is run.
- **Strong Static Typing** – JavaScript is not strongly typed. TypeScript comes with an optional static typing and type inference system through the TLS (TypeScript Language Service). The type of a variable, declared with no type, may be inferred by the TLS based on its value.
- TypeScript **supports type definitions** for existing JavaScript libraries. TypeScript Definition file (with **.d.ts** extension) provides definition for external JavaScript libraries. Hence, TypeScript code can contain these libraries.
- TypeScript **supports Object Oriented Programming** concepts like classes, interfaces, inheritance, etc.

1. Calculator.ts:

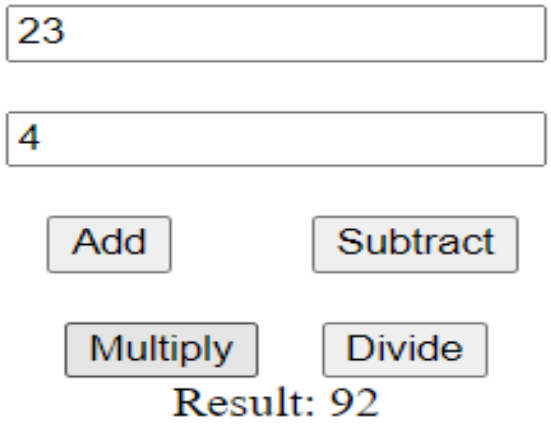
Html code:

```
<!DOCTYPE html>
<html>
<head>
   <title>Calculator</title>
</head>
<body align="center" >
   <br><br><br><br><br><br><br>
   <input id="num1" type="number" placeholder="Number 1"><br><br>
   <input id="num2" type="number" placeholder="Number 2"><br><br>
    <button id="add">Add</button>          
   <button id="subtract">Subtract</button><br><br>
   <button id="multiply">Multiply</button>     
   <button id="divide">Divide</button>
   <div id="result"></div>

   <script src="app.js"></script>
</body>
</html>
```

TypeScript code:

```
window.onload = () => {
    const num1 = document.getElementById("num1") as HTMLInputElement;
    const num2 = document.getElementById("num2") as HTMLInputElement;
    const addButton = document.getElementById("add") as HTMLButtonElement;
    const subtractButton = document.getElementById("subtract") as
HTMLButtonElement;
    const multiplyButton = document.getElementById("multiply") as HTMLButtonElement;
    const divideButton = document.getElementById("divide") as HTMLButtonElement;
    const resultDiv = document.getElementById("result");

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

    function subtract(a: number, b: number): number {
        return a - b;
    }

    function multiply(a: number, b: number): number {
        return a * b;
    }

    function divide(a: number, b: number): string | number {
        if (b === 0) {
            return "Cannot divide by zero";
        }
        return a / b;
    }

        function performOperation(operation: (a: number, b: number) => string | number) {
        const a = parseFloat(num1.value);
        const b = parseFloat(num2.value);
        const result = operation(a, b);
        resultDiv!.innerText = `Result: ${result}`;
    }
```

```
    addButton.addEventListener("click", () => performOperation(add));
    subtractButton.addEventListener("click", () => performOperation(subtract));
    multiplyButton.addEventListener("click", () => performOperation(multiply));
    divideButton.addEventListener("click", () => performOperation(divide));
};
```



2. Inheritance.ts:

```
interface Animal {
    makeSound(): void;
}

class Creature {
    protected name: string;
    private creatureType: string;
    constructor(name: string, creatureType: string) {
        this.name = name;
        this.creatureType = creatureType;
    }

    public display(): void {
        console.log(`This creature is a ${this.name}. It's a ${this.creatureType}.`);
    }

    protected getCreatureType(): string {
```

```
        return this.creatureType;
    }
}

class Dog extends Creature implements Animal {
    constructor(name: string) {
        super(name, "Dog");
    }

    public makeSound(): void {
        console.log("Woof! Woof!");
    }

    public showCreatureType(): void {
        console.log(`The creature type is ${this.getCreatureType()}.`);
    }
}

let myDog: Dog = new Dog("Golden Retriever");
myDog.display();
myDog.makeSound();
myDog.showCreatureType();
```

```
PS C:\Users\Kaushik Kotian\OneDrive\Desktop\calculator_ts> node inher.js
This creature is a Golden Retriever. It's a Dog.
Woof! Woof!
The creature type is Dog.
```