# Experiment No: 3

**Aim:** Small code snippets for programs like Hello World, Calculator using TypeScript.

## Theory:

TypeScript is a superset of JavaScript which primarily provides optional static typing, classes and interfaces. One of the big benefits is to enable IDEs to provide a richer environment for spotting common errors as you type the code.

TypeScript doesn't run in the browser. The code written in typescript is compiled toJavaScript, which then runs in the browser.

TypeScript is open source. It was designed by Anders Hejlsberg (the lead architect of C#) at Microsoft.

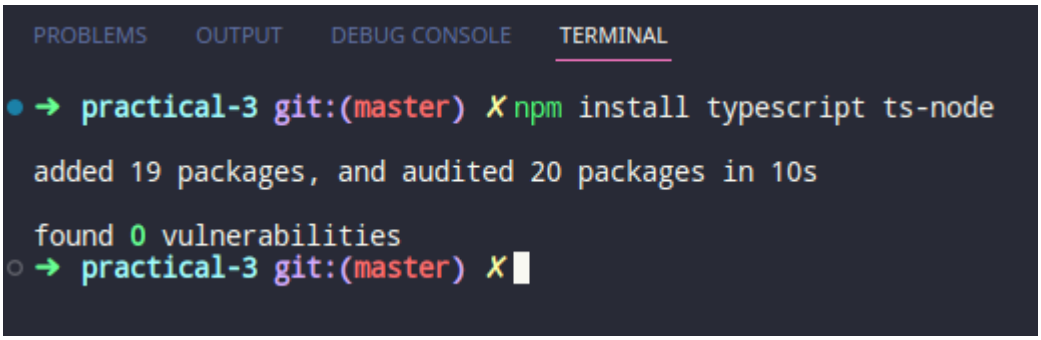
**Features of TypeScript:**

1. The code written in typescript is compiled to the JavaScript for the purpose of execution.
2. As we already discussed that the code written in typescript is compiled to the JavaScript which can reuse all of the existing JavaScript frameworks, tools, and libraries.
3. As TypeScript is a superset of JavaScript so any valid .js file can be renamed to .ts andcompiled with other TypeScript files.
4. TypeScript is portable i.e. it can run on any environment that JavaScript runs on.
5. TypeScript provides optional static typing. JavaScript is dynamically typed. This means JavaScript does not know what type a variable is until it actually instantiated at run-time. This also means that it may be too late. TypeScript adds type support to JavaScript. TypeScript makes typing a bit easier and alot less explicit by the usage of type inference. For example: var msg = "hello" in TypeScript is the same as var msg : string = "hello". TLS (TypeScript Language Service) provides the facility to identify the type of the variable based on its value if it is declared with no type.
6. TypeScript supports Object Oriented Programming concepts like classes, object, interfaces and inheritance etc.
7. TypeScript Definition file with .ts extension provides definition for external JavaScriptlibraries.
8. TypeScript provides strict null checks. TypeScript compiler will not allow undefined to be

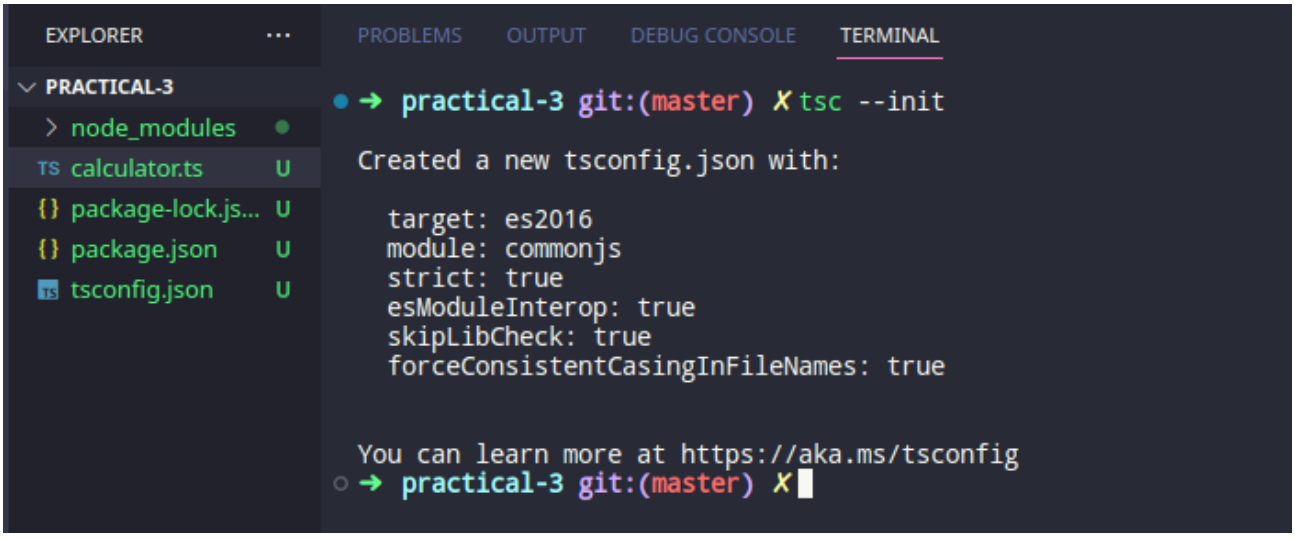
assigned to a variable unless you explicitly declare it to be of nullable type.
For example, let num1 : number = undefined will result in a compile error. This fits perfectly with type theory, since undefined is not a number. One can define x to be a sum type of number and undefined to correct this: let num1 : number | undefined = undefined.

**Steps:**

**1. Install TypeScript**




**2. Generate a tsconfig.json file**




**To run a TypeScript file, execute following commands :**

- *tsc calculator.ts && node calculator.js*

  OR

- *ts-node calculator.ts*

**Code:**

```
import { clear } from "console";
import Prompt from "prompt-sync";
("prompt-sync");
const prompt = Prompt({ sigint: true });
clear();

const add = (num1: number, num2: number): number => num1 + num2; const
sub = (num1: number, num2: number): number => num1 - num2; const mul =
(num1: number, num2: number): number => num1 * num2; const div = (num1:
number, num2: number): number => num1 / num2; const power = (num:
number, pow: number): number => num ** pow;

const menu = (): string => {
 const menuString = `
----------------------
 Options
----------------------
1. Add
2. Subtract
3. Multiply
4. Divide
0. Exit
5. Power`;
 console.log(menuString);
 const choice = prompt("choice:");
 console.log("----------------------");

 switch (choice) {
 case "1": {
 let a: number = parseInt(prompt("Number 1: ") as string);  let b: number =
parseInt(prompt("Number 2: ") as string);  console.log("Adding.....");
 console.log(`${a} + ${b} = ${add(a, b)}`);
 break;
 }
 case "2": {
```

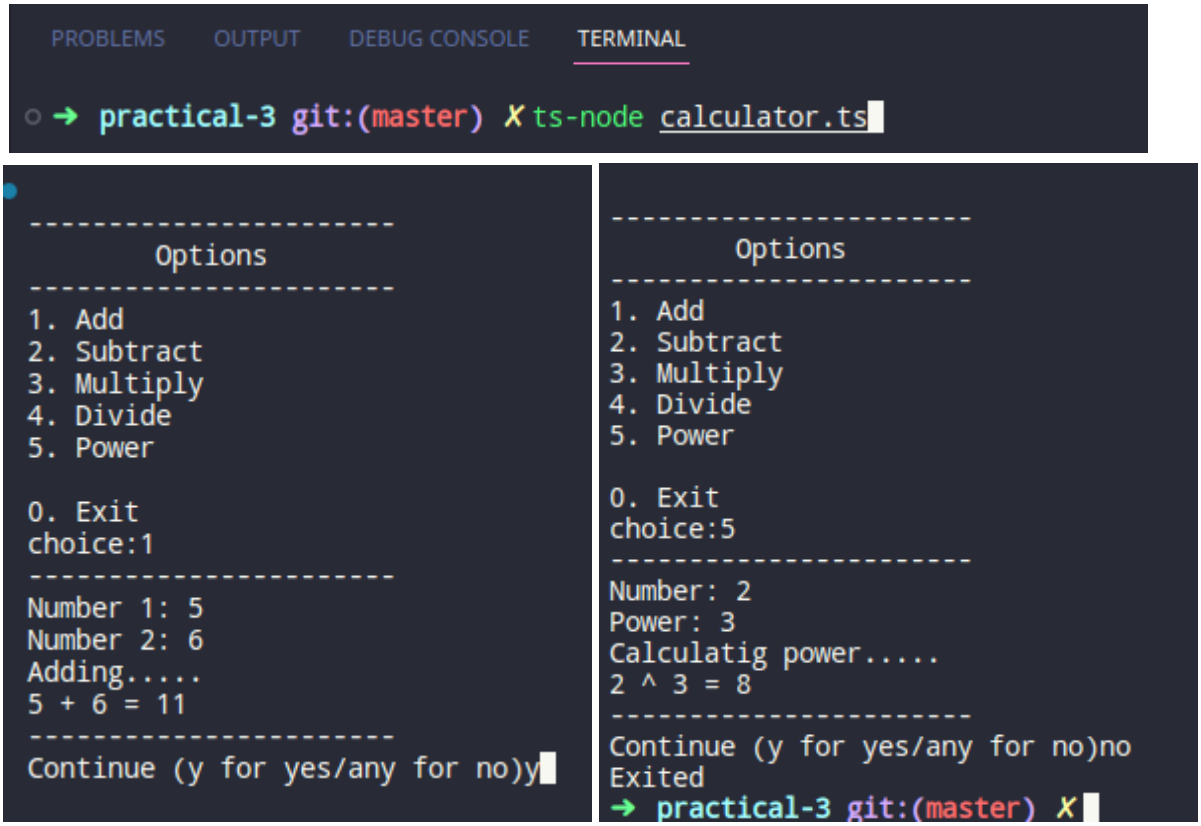
```
 let a: number = parseInt(prompt("Number 1: ") as string);  let b: number =
parseInt(prompt("Number 2: ") as string);  console.log("Subtracting.....");
 console.log(`${a} - ${b} = ${sub(a, b)}`);
 break;
 }
 case "3": {
 let a: number = parseInt(prompt("Number 1: ") as string);
 let b: number = parseInt(prompt("Number 2: ") as string);
console.log("Multiplying.....");
 console.log(`${a} * ${b} = ${mul(a, b)}`);
 break;
 }
 case "4": {
 let a: number = parseInt(prompt("Number 1: ") as string);  let b: number =
parseInt(prompt("Number 2: ") as string);  console.log("Divding.....");
 console.log(`${a} / ${b} = ${div(a, b)}`);
 break;
 }
 case "5": {
 let num: number = parseInt(prompt("Number: ") as string);  let pow: number
= parseInt(prompt("Power: ") as string);  console.log("Calculatig power.....");
 console.log(`${num} ^ ${pow} = ${power(num, pow)}`);  break;
 }
 case "0": {
 console.log("Exiting.......");
 }
 default:
 console.log("Invalid choice!");
 }
 console.log("----------------------");

 return prompt("Continue (y for yes/any for no)");
};

while(menu().toLowerCase() == "y"){
 clear();
```

```
}
console.log("Exited");
```

**Output:**




**Conclusion**:

Writing small programs like "Hello World" and calculators using TypeScript helps in understanding type safety and error handling. TypeScript provides additional features over JavaScript, making code more structured, readable, and scalable. By implementing basic examples, developers gain confidence in working with strongly typed languages.