**Experiment No.: 01**

**Title:** Demonstrate various programming concept using Typescript.

**Objective:**

1. To develop basic building blocks of Typescript

**Theory:**

TypeScript is a programming language. It is a superset of JavaScript, which means it builds on top of JavaScript by adding features like static typing, interfaces, and other enhancements to help developers write more robust and scalable code.

JavaScript is a loosely typed language. It can be difficult to understand what types of data are being passed around in JavaScript. But, TypeScript allows specifying the types of data being passed around within the code, and has the ability to report errors when the types don't match. For example, TypeScript will report an error when passing a string into a function that expects a number. JavaScript will not.

TypeScript is transpiled into JavaScript using a compiler. TypeScript has an official compiler which can be installed through npm as:

*npm install –g typescript*

To compile and execute the typescript file having name as test.ts file following command can be used respectively.

* *tsc test.ts*
* *node test*

Following are the different building block of typescript.

**1. Types**

There are three main primitives type in TypeScript.

* boolean - true or false values
* number - whole numbers and floating point values
* string - text values like "TypeScript Rocks"

There are two ways to define the type of the variable namely: Explicit and Implicit.

* Explicit type assignment is easier to read and more intentional. Example: let firstName: string = "Dylan";
* Implicit - TypeScript will "guess" the type, based on the assigned value:

let firstName = "Dylan";

Following are the few more types:

* any is a type that disables type checking and effectively allows all types to be used.
* undefined and null are types that refer to the typescript primitives undefined and null respectively.

**2. Function:**

TypeScript has a specific syntax for typing function parameters and return values.

function name(parameter1, parameter2, parameter3):void {

// code to be executed

}

**3. Interface**

Interface is a structure that defines the contract in application. It defines the syntax for classes to follow. Classes that are derived from an interface must follow the structure provided by their interface. The TypeScript compiler does not convert interface to JavaScript. It uses interface for type checking. This is also known as "duck typing" or "structural subtyping".

An interface is defined with the keyword interface and it can include properties and method declarations using a function or an [arrow function](https://www.tutorialsteacher.com/typescript/arrow-function).

interface interface\_name { }

Interface in TypeScript can be used to define a type and also to implement it in the class. TypeScript interface is also used to define a type of a function. This ensures the function signature. An interface can also define the type of an array where you can define the type of index as well as values.

**4. Interface & Inheritance-**

An interface can be extended by other interfaces. In other words, an interface can inherit from other interface. Typescript allows an interface to inherit from multiple interfaces. Use the extends keyword to implement inheritance among interfaces.

Single Interface Inheritance Syntax

Child\_interface\_name extends super\_interface\_name

Multiple Interface Inheritance Syntax

Child\_interface\_name extends super\_interface1\_name, super\_interface2\_name, … , super\_interfaceN\_name

**5. Class**

A class is a blueprint for creating objects. A class encapsulates data for the object. Typescript gives built in support for class. Use the class keyword to declare a class in TypeScript. The syntax is given below –

class class\_name {

//class scope

}

A class can include the following:

* Constructor
* Properties
* Methods

An object is an instance which contains set of key value pairs. The values can be scalar values or functions or even array of other objects. An object of the class can be created using the [new keyword](https://www.tutorialsteacher.com/javascript/new-keyword-in-javascript).

**6. Modules**

In TypeScript, modules are used to organize code into reusable units. They allow you to encapsulate functionality and declare dependencies between different parts of your code. Here's a small example to demonstrate how modules work in TypeScript:

Let's say we have two files:

mathFunctions.ts: This file contains some mathematical functions.

app.ts: This is our main application file where we want to use the functions defined in mathFunctions.ts.

// mathFunctions.ts

// Function to add two numbers

export function add(a: number, b: number): number {

return a + b;

}

// Function to subtract two numbers

export function subtract(a: number, b: number): number {

return a - b;

}

In this file, we have two functions add and subtract, and we're using the export keyword to make them accessible outside of this module.

Now, let's use these functions in app.ts:

// app.ts

// Importing functions from mathFunctions.ts

import { add, subtract } from './mathFunctions';

// Using the imported functions

console.log(add(5, 3)); // Output: 8

console.log(subtract(10, 4)); // Output: 6

In app.ts, we import the add and subtract functions from mathFunctions.ts using the import keyword. We specify the path to the module using a relative path ('./mathFunctions'), assuming that both files are in the same directory.

When you compile and run app.ts, it will use the functions defined in mathFunctions.ts.

This is a simple example of how modules work in TypeScript. They allow you to organize your code into separate files and reuse functionality across your application.

**Key Concept:** typescript, npm

**Steps:**

1. Install nodejs on your machine

2. Use npm to install typescript compiles.

3. Write typescript program with different building blocks and save with ts extension.

4. Compile the file with tsc test.ts

5. Run the program with node test