Name of Student: Pushkar Sane		
Roll Number: 45	Lab Assignment Number: 8	
Title of Lab Assignment: TypeScript installation, Environment Setup, Programs on decision making / functions / class & object.		
DOP: 16-10-2023	DOS: 20-10-2023	
CO Mapped: CO4	PO Mapped: PO3, PO5, PSO1, PSO2	Signature:

Practical No. 8

Aim: TypeScript installation, Environment Setup, Programs on decision making/functions/class & object.

Description:

TypeScript Installation, Environment Setup, and How TypeScript Works:

TypeScript is a statically typed superset of JavaScript that provides optional static typing, enhanced tooling, and other features to improve the development experience. Here's a comprehensive note on installing TypeScript, setting up your development environment, and understanding how TypeScript works.

1. **TypeScript Installation:**
   To install TypeScript, you need to have Node.js and npm (Node Package Manager) on your system.
   Follow these steps:
   a. Install Node.js: Download and install Node.js from the [official Node.js website](https://nodejs.org/).
      Node.js comes with npm, which is required to install TypeScript.
   b. Install TypeScript:
      Open your command-line interface (CLI) and run the following command to install TypeScript globally on your system:
      ```
      bash
      npm install -g typescript
      ```
      This command installs TypeScript globally, making it available for use on your system.

2. **Environment Setup:**
   After installing TypeScript, you can set up your development environment to start writing TypeScript code.
   a. Create a Project Directory: Choose or create a directory where you want to work on your TypeScript project. You can do this through your file explorer or with the following command.

```bash
mkdir my-typescript-project
cd my-typescript-project
```

b. Initialize a TypeScript Project: To configure your TypeScript project, create a `tsconfig.json` configuration file using the following command.

```bash
tsc --init
```

This command generates a basic `tsconfig.json` file in your project directory. You can modify this file to customize TypeScript settings for your project.

c. Choose a Code Editor or IDE: Select a code editor or integrated development environment (IDE) that supports TypeScript. Popular choices include Visual Studio Code, WebStorm, and Sublime Text. Download and install your preferred editor.

Here's an overview of how TypeScript works and its key features:

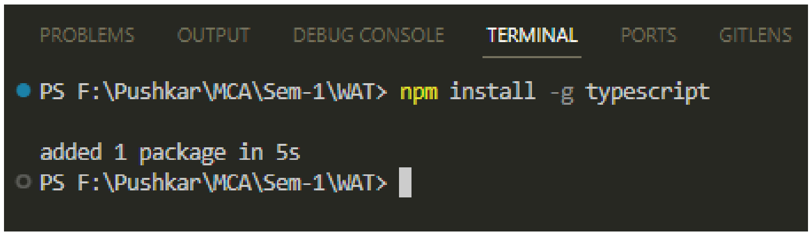
1. **TypeScript Code:**
   a. You write TypeScript code in `.ts` files. TypeScript supports modern JavaScript syntax, including ES6 and ESNext features.
   b. You can define variable types and use type annotations to provide information about the data types of variables, parameters, and function return values.
2. **Compilation:**
   a. TypeScript code is not directly understood by browsers or Node.js. To run TypeScript code, it must be transpiled to plain JavaScript.
   b. The TypeScript compiler, `tsc`, is used to transpile `.ts` files into `.js` files.
   c. The transpilation process involves checking for type errors, type inference, and removing TypeScriptspecific syntax. The output is JavaScript code that can be executed in a JavaScript runtime environment.
3. **Type Checking:**
   a. TypeScript provides optional static typing, allowing you to specify data types for variables and function parameters.
   b. The TypeScript compiler performs type checking during compilation. It flags type-related errors and provides warnings, helping to catch potential issues early in the development process.

4. **Tooling Support:**
    a. TypeScript offers enhanced tooling support, including autocompletion, refactoring, and navigation features in code editors and IDEs.
    b. Development environments such as Visual Studio Code provide built-in TypeScript support, making it easier to write, debug, and manage TypeScript projects.
5. **Execution:**
    a. The transpiled JavaScript code can be executed in any JavaScript environment, whether it's a web browser or Node.js.
    b. The runtime behavior of the code is the same as if it were originally written in JavaScript.

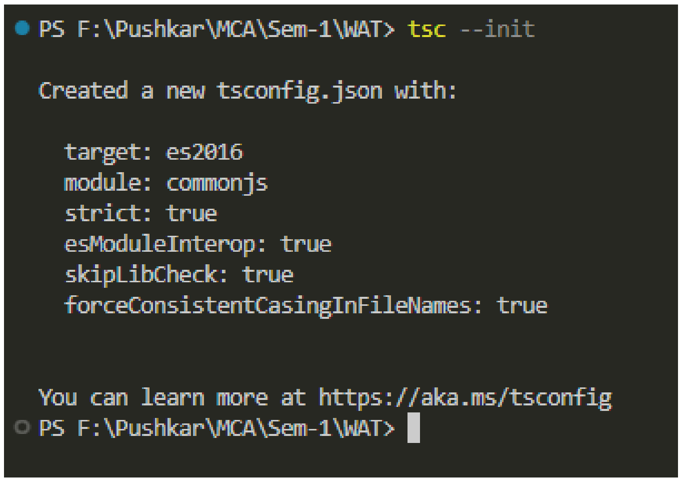
In conclusion, TypeScript is a valuable tool that enhances JavaScript development by providing static typing, enhanced tooling, and type checking while maintaining compatibility with JavaScript. By following the installation and environment setup steps, you can effectively work with TypeScript in your projects, catching errors early and improving code quality.

Code:

TypeScript installation, Environment Setup:



```
PS F:\Pushkar\MCA\Sem-1\WAT> npm install -g typescript

added 1 package in 5s
PS F:\Pushkar\MCA\Sem-1\WAT>
```

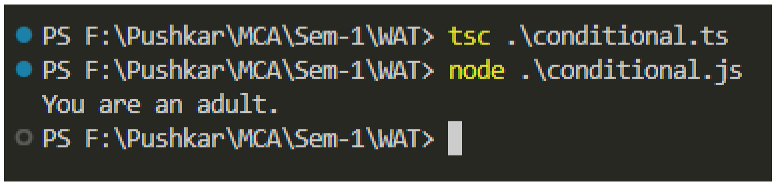


```
PS F:\Pushkar\MCA\Sem-1\WAT> tsc --init

Created a new tsconfig.json with:

  target: es2016
  module: commonjs
  strict: true
  esModuleInterop: true
  skipLibCheck: true
  forceConsistentCasingInFileNames: true


You can learn more at https://aka.ms/tsconfig
PS F:\Pushkar\MCA\Sem-1\WAT>
```

1. **Program on decision making**
    a. **conditional.ts**

```
let age: number = 18;
if (age >= 18) {
   console.log("You are an adult.");
} else {
   console.log("You are a minor.");
}
```

Output:

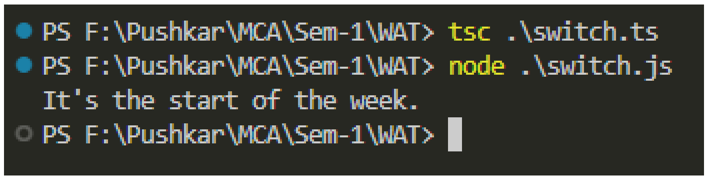


```
PS F:\Pushkar\MCA\Sem-1\WAT> tsc .\conditional.ts
PS F:\Pushkar\MCA\Sem-1\WAT> node .\conditional.js
You are an adult.
PS F:\Pushkar\MCA\Sem-1\WAT>
```

    b. **switch.ts**

```
let day: string = "Monday";
switch (day) {
   case "Monday":
      console.log("It's the start of the week.");
      break;
   case "Friday":
      console.log("It's almost the weekend!");
      break;
   default:
      console.log("It's a regular day.");
}
```

Output:



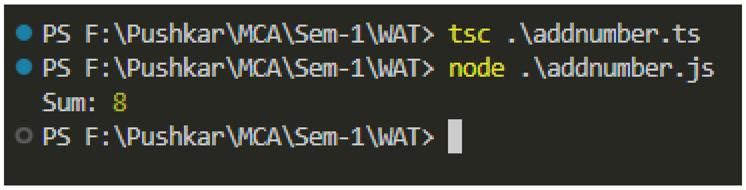
```
PS F:\Pushkar\MCA\Sem-1\WAT> tsc .\switch.ts
PS F:\Pushkar\MCA\Sem-1\WAT> node .\switch.js
It's the start of the week.
PS F:\Pushkar\MCA\Sem-1\WAT>
```

2. Function Programs:

a. addnumbers.ts

```
function add(a: number, b: number): number {
   return a + b;
}
let result: number = add(5, 3);
console.log("Sum:", result);
```

Output:

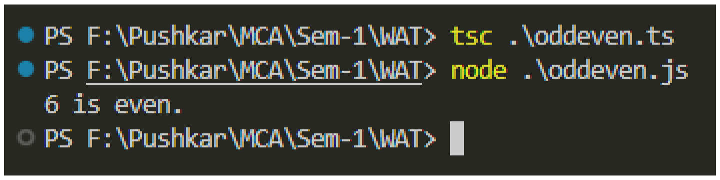


```
● PS F:\Pushkar\MCA\Sem-1\WAT> tsc .\addnumber.ts
● PS F:\Pushkar\MCA\Sem-1\WAT> node .\addnumber.js
  Sum: 8
○ PS F:\Pushkar\MCA\Sem-1\WAT>
```

b. oddeven.ts

```
function isEven(num: number): boolean {
   return num % 2 === 0;
}
let numberToCheck: number = 6;
if (isEven(numberToCheck)) {
   console.log(numberToCheck + " is even.");
} else {
   console.log(numberToCheck + " is odd.");
}
```

Output:



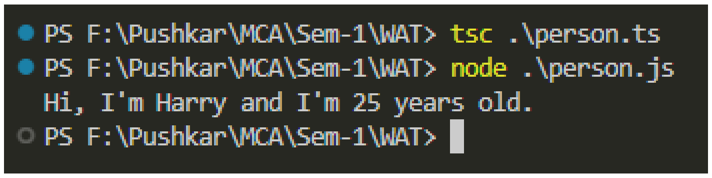
```
● PS F:\Pushkar\MCA\Sem-1\WAT> tsc .\oddeven.ts
● PS F:\Pushkar\MCA\Sem-1\WAT> node .\oddeven.js
  6 is even.
○ PS F:\Pushkar\MCA\Sem-1\WAT>
```

3. Class and Object programs:

a. person.js

```
class Person {
  name: string;
  age: number;
  constructor(name: string, age: number) {
    this.name = name;
    this.age = age;
  }
  introduce(): string {
    return `Hi, I'm ${this.name} and I'm ${this.age} years old.`;
  }
}
let person1: Person = new Person("Alice", 30);
console.log(person1.introduce());
```

Output:



```
PS F:\Pushkar\MCA\Sem-1\WAT> tsc .\person.ts
PS F:\Pushkar\MCA\Sem-1\WAT> node .\person.js
Hi, I'm Harry and I'm 25 years old.
PS F:\Pushkar\MCA\Sem-1\WAT>
```

Conclusion:

By setting up TypeScript and leveraging its features, you can write more robust and maintainable JavaScript code while benefiting from a richer development experience. TypeScript's ability to catch errors early and enhance tooling support makes it a valuable addition to your development toolkit.