Introduction:

TypeScript is a superset of JavaScript that adds extra features like static typing, interfaces, enums, and more. Essentially, TypeScript is JavaScript with additional syntax for defining types, making it a powerful tool for building scalable and maintainable applications.

* Static typing allows you to define variable types and helps catch errors before running the code.
* TypeScript is compiled into JavaScript, ensuring it works in all JavaScript environments.
* It’s widely used for web development on both the client and server sides.
* Developed by Microsoft, TypeScript is trusted by companies like Google, Slack, Airbnb, and Asana.

Typescript frameworks:

1. Angular
2. Adonisjs
3. Nestjs
4. FoalTS
5. TypeORM
6. FeatherJs

**Why Learn TypeScript?**

TypeScript is becoming the preferred language for modern web development due to its advantages over JavaScript:

* **Improved Code Quality:** TypeScript adds type checking to JavaScript, which helps catch mistakes early and makes the code safer to use.
* **Better Scalability:** TypeScript is used to manage and grow large projects. It makes it easier to maintain complex codebases over time.
* **Works Well with JavaScript**: TypeScript is a superset of JavaScript, which means you can use both in the same project. You don’t need to rewrite your existing JavaScript code to start using TypeScript.
* **Better Developer Tools:** TypeScript works well with code editors and tools. It offers features like autocompletion, debugging, and type checking, which help developers write code faster and with fewer errors.
* **Works With Popular Frameworks:** TypeScript is commonly used with frameworks like React, Angular, and Vue, which improves type safety and the overall developer experience..
* **Catch Bugs Early:**TypeScript helps find bugs at compile time (before the code runs). This saves time by catching errors early.
* **More Job Opportunities:**As TypeScript becomes more popular, it opens more job opportunities, especially in companies working with large-scale web applications.

Installation and Uninstallation

1. **Open Command Prompt**: Go to the Start menu and open the command prompt
2. **Use NPM to Install TypeScript**: Run the following command to install TypeScript globally

npm install --global typescript

1. **Verify Installation**: Check the installed version of TypeScript by running

**tsc** –v

### **Uninstall TypeScript**

npm uninstall --global typescript

## Creating a Simple TypeScript Project

Let’s create a simple “Hello World” project in TypeScript to verify the installation and understand the basic workflow.

### Step 1: Create a Project Folder

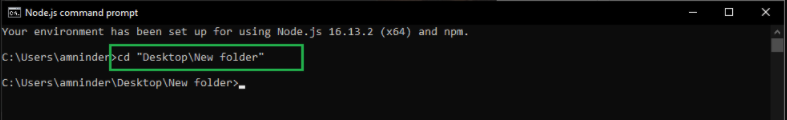
**1. Create Folder:**Create a folder on your desktop and name it, e.g., typescript-hello-world.

**2. Create a TypeScript File:**Inside this folder, create a file named main.ts.

### **Step 2: Initialize TypeScript Project**

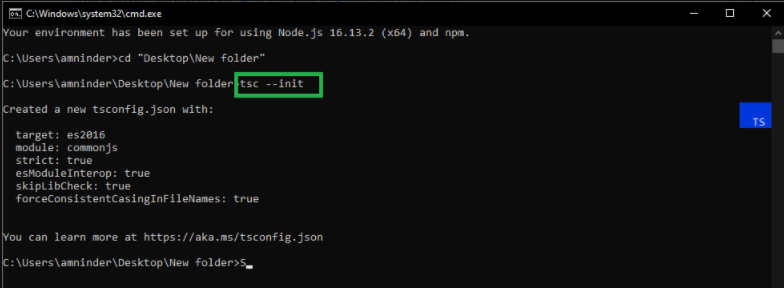
1. **Open Command Prompt**: Open the Node.js command prompt and navigate to the project folder:

cd path/to/typescript-hello-world



**2**. **Initialize TypeScript Configuration:** Run the following command to create a tsconfig.json file:

tsc --init



This file contains all the configurations for the TypeScript compiler.

### **Step 3: Write TypeScript Code**

**1. Open main.ts:** Open the main.ts file and add the following code.

console.log("Hello, TypeScript!");

**2. Save the File:** Save the main.ts file.

### **Step 4: Compile TypeScript Code**

**1. Compile TypeScript:**Run the TypeScript compiler with the file name as an argument:

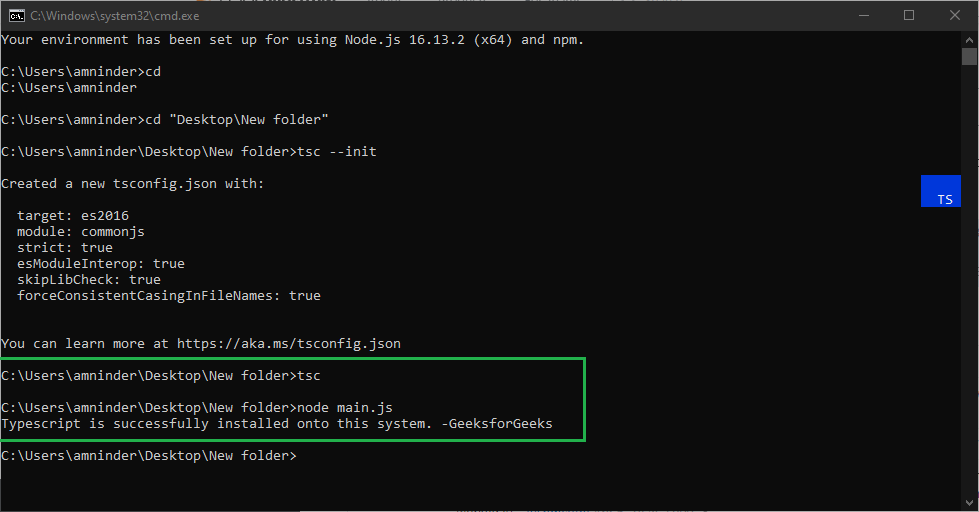
tsc

### Step 5: Run the JavaScript Code

**1. Run with Node.js:** Execute the compiled JavaScript file using Node.js

node main.js

**Output:**



*Typescript hello world*

In this guide, we covered the installation and basic usage of TypeScript. By setting up TypeScript, you can catch errors early in the development process, improving code quality and reducing runtime errors.

## Key Features of TypeScript

### 1. Static Type Checking (Optional)

TypeScript allows you to check and assign types to variables, parameters, and function return values. While this step requires a little more effort, it significantly improves code quality. Optional static typing helps prevent bugs and makes your code more readable.

### 2. Class-Based Objects

One of TypeScript’s standout features is its support for **classes**. Unlike JavaScript’s prototype-based approach, TypeScript lets you write true object-oriented code. You can create classes, define constructors, and use inheritance and access modifiers (public, private, protected).

### 3. Modularity

TypeScript promotes modularity. By using modules, you can organize your code into smaller, reusable pieces. This modularity enhances maintainability and collaboration among team members.

### 4. ES6 Features

TypeScript embraces [**ECMAScript 6 (ES6)**](https://www.geeksforgeeks.org/introduction-to-es6/) features. If you’re familiar with ES6 syntax (arrow functions, template literals, destructuring, etc.), you’ll feel right at home with TypeScript.

### 5. Syntactical Sugaring

TypeScript’s syntax is closer to that of high-level languages like [Java](https://www.geeksforgeeks.org/java/). It’s like a sweetener for developers—more concise and expressive.

Variables in TypeScript are used to store data values, acting as named memory locations that can hold numbers, strings, booleans, or other types of data.

* Variables can be declared using let, const, or var depending on the use case.
* They provide type safety by allowing you to define specific data types like string, number, or boolean.
* TypeScript supports both explicitly typed and type-inferred variable declarations.

#### Declare Type and Value in a Single Statement

**let** name: string = 'Amit';

**const** age: number = 25;

Variable Scopes in TS

1.Local Scope

2. Global Scope

3. Class Scope

**let** globalVar: number = 10;

**class** Geeks {

**private** classVar: number = 11;

assignNum(): **void** {

**let** localVar: number = 12;

console.log('Local Variable: ' + localVar);

}

}

console.log('Global Variable: ' + globalVar);

**let** obj = **new** Geeks();

obj.assignNum();

Datatype of TS(same as JS)

Annotations in TS

**const** str: string = "GeeksforGeeks";

**const** num: number = 6;

**const** arr: (number | string)[] =

["GFG", "TypeScript", 500, 20];

console.log(**typeof** str);

console.log(**typeof** num);

console.log(arr);

**function** greet(name: string): string {

**return** `Hello, **${**name**}**!`;

}

**const** person: { name: string; age: number } = {

name: "Alice",

age: 30

};

class Rectangle {

width: number;

height: number;

constructor(width: number, height: number) {

this.width = width;

this.height = height;

}

area(): number {

return this.width \* this.height;

}

}

Inference  
TypeScript's type inference automatically determines the types of variables, function return values, objects, and arrays based on their assigned values and usage.

* This feature reduces the need for explicit type annotations, simplifying code while maintaining type safety.
* By analyzing the context and initial values, TypeScript ensures that variables and functions operate with consistent and expected types throughout the codebase.

let age = 25;

let name = "John";

console.log(`Age: ${age}`);

console.log(`Name: ${name}`);

**let** x = 100;

x = "Hello";

TypeScript throws a compilation error

How can you explicitly define a variable with multiple possible types?

let value: multiple<number, string> = 42;

null and undefined cannot be assigned to variables of other types unless explicitly allowed

<https://www.geeksforgeeks.org/typescript-enums/>

# TypeScript Literal Types

TypeScript's literal types allow developers to specify exact values for variables, function parameters, or properties, enhancing type safety by ensuring variables can only hold predefined values.

* Allow variables to have specific, exact values.
* Enhance code reliability by restricting permissible values.

type Direction = "Up" | "Down" | "Left" | "Right";

let move: Direction;

move = "Up"; // Valid assignment

// move = "Forward"; // Error: Type '"Forward"' is not assignable to type 'Direction'.

move = "Up"; // No error  
move = "Forward"; // Compile-time error

type DiceRoll = 1 | 2 | 3 | 4 | 5 | 6;

function rollDice(): DiceRoll {

return 4; // Valid return value

// return 7; // Error: Type '7' is not assignable to type 'DiceRoll'.

}

rollDice(); // Returns 4 without error  
rollDice(); // Returning 7 causes a compile-time error

type Success = true;

function operation(): Success {

return true; // Valid return value

// return false; // Error: Type 'false' is not assignable to type 'true'.

}

**operation();** // Returns true without error  
**operation();** // Returning false causes a compile-time error