VARIABLES

In TypeScript, a variable is a named space in the memory that stores data that can be referenced and manipulated in a program. Variables in TypeScript are similar to those in JavaScript but come with the added advantage of static type checking, which helps catch errors at compile time rather than at runtime.

1. **Variable Declaration**: Variables in TypeScript can be declared using the let, const, and var keywords.
   * let is used for variables that can be reassigned.
   * const is used for variables that are constant and cannot be reassigned.
   * var is similar to let but has function scope instead of block scope.
2. **Type Annotations**: TypeScript allows you to specify the type of the variable at the time of declaration. This is called a type annotation.

let num: number = 10;

const str: string = "Hello, TypeScript";

1. **Type Inference**: If you do not explicitly provide a type, TypeScript will infer the type based on the assigned value.

let num = 10; // TypeScript infers num as number

const str = "Hello, TypeScript"; // TypeScript infers str as string

1. **Scope**: The scope of variables in TypeScript depends on how they are declared.
   * let and const have **block scope**, meaning they are only accessible within the block where they are defined.
   * var has **function scope**, meaning it is accessible within the function where it is defined.
2. **Reassignment**:
   * Variables declared with let can be reassigned.

let num = 10;

num = 20; // Valid

* + Variables declared with const cannot be reassigned.

const num = 10;

num = 20; // Error: Assignment to constant variable

1. **Initialization**:
   * Variables declared with let and const must be initialized before they are used.

let num; // Allowed, but num will have the type 'any'

num = 10;

const str; // Error: const declarations must be initialized

1. **Types**: TypeScript supports a wide range of types including **primitive types (**like **number, string, boolean),**

**complex types** (like arrays and objects),

and **user-defined types** (like interfaces and enums).

Here are some examples:

// Using let and const with type annotations

let age: number = 25;

const name: string = "Alice";

// Type inference

let isStudent = true; // inferred as boolean

// Array

let numbers: number[] = [1, 2, 3, 4, 5];

// Object

let person: { name: string, age: number } = { name: "John", age: 30 };

// Function with typed parameters and return type

function greet(name: string): string {

return `Hello, ${name}`;

}

By using TypeScript's type system, you can write more robust