Lecture 9:
Intro to TypeScript

Features of TypeScript

- **Static Typing -** Explicitly defining variables, function parameters and return values
- **Type Inference –** Automatically infers types based on how variables are initialized
- **Interfaces and Type Aliases –** Allows you to define custom types or structure for objects, improving code structure and understanding
- **Classes and Object-Oriented Programming –** TypeScript makes OOP features like inheritance, access modifiers and abstract classes more powerful
- **Generics –** They allow for writing reusable components and functions
- **Compile-Time Checking –** Errors can be caught at compile time, reducing runtime issues.
- **Compatibility with JavaScript –** You can use JavaScript code in a TypeScript file. It contains full compatibility with JavaScript libraries and frameworks.

Why Use TypeScript?

Better Code Quality

Catch errors early and enforce a consistent structure

Readability and Maintainability

Explicit types and interfaces make code easier to understand

Refactoring and Tooling

IDEs provide autocompletion, navigation and refactoring tools

Scalability

Type system makes large codebases easier to manage

Supports modern JavaScript

Supports new JavaScript features, ensuring you can use latest capabilities.

Basic Types

Boolean

Represents true/false values

Number

Represents numeric values, both integers and floating points

String

Represents textual data in quotes

Array

A collection of elements of the same type

Tuple

Similar to an array, but with a fixed number of elements, each with a specific type

Basic Types

Enum

A way to define a set of named constants, improving code readability

Any

A flexible type that can hold any value, useful when the type of data is unknown

Void

Typically used for functions that don't return a value

Null & Undefined

Represent the absence of value. Null is explicitly assigned, undefined is for uninitialized variables

Never

Represents values that never occur, such as a function that never returns or always throws an error

Object

Represents non-primitive types like arrays, functions, or objects

What is Type Assertion

- Informs the TypeScript compiler to treat a value as a specific type
- Does not change runtime behavior
- Useful with **any** type or when the compiler cannot infer the correct type

Syntax for Type Assertion

1. **Angle – bracket Syntax**

```
let someValue: any = "this is a string";

let strLength: number = (<string>someValue).length;
```

2. **as syntax**

```
let strLength: number = (someValue as string).length;
```

Variable Declarations

var

- Function-scoped or globally scoped
- Can be redeclared and updated
- Allows hoisting (accessible before declaration)

let

- Block-scoped (local scope)
- Cannot be redeclared but can be updated
- No hoisting (must be declared before use)

const

- Block-scoped (local scope)
- Cannot be redeclared or updated
- Must be initialized during declaration

Scopes

Global Scope

- Variables accessible throughout the entire code
- Declared outside any function or class

Class Scope

- Variables accessible within a class
- Declared inside a class but outside any methods

Local Scope

- Variables accessible only within a specific block or function
- Declared inside a function, loop, or block of code

Destructuring

Array Destructuring

- Unpacks elements from an array into individual variables
- Provides a concise way to assign values from an array

Object Destructuring

- Unpacks properties from an object into individual variables
- Allows direct access to object properties with simpler syntax

Function Declarations

Parameter Type

- Specifies the type of arguments a function accepts
- Ensures correct types are passed to the function

Return Type

- Defines the type of value a function returns
- Helps TypeScript catch errors if a different type is returned

Function Parameters

Optional Parameter

- Parameters that are not mandatory
- Denoted by a **?** after the parameter name

Default Parameter

- Assigns a default value to a parameter if no argument is provided
- Ensures functions behave correctly with missing arguments

Rest Parameters

- Collects multiple arguments into a single array
- Allows a function to accept an indefinite number of arguments