**Guide - TypeScript**

1. Introduction to TypeScript
   1. TypeScript is a statically typed superset of JavaScript that adds optional static typing to the language. Developed by Microsoft, it compiles to plain JavaScript, allowing it to run in any environment that JavaScript runs in.
2. Why Use TypeScript
   1. Static Typing: Detects errors at compile-time rather than runtime.
   2. Enhanced IDE Support: Features like autocompletion, type checking, and refactoring.
   3. Improved Readability and Maintainability: Makes the codebase easier to understand and maintain.
   4. Large-scale Development: Facilitates the management of complex applications with multiple developers.
3. Installation Instructions for TypeScript Compiler (tsc)
   1. To install TypeScript, use npm (Node Package Manager):
      1. npm install -g typescript
         1. This installs the TypeScript compiler globally on your system.
4. Setting Up a New TypeScript Project
   1. Initialize a new project:
      1. mkdir my-typescript-project
      2. cd my-typescript-project
      3. npm init -y
5. Install TypeScript locally:
   1. npm install --save-dev typescript
   2. Create a tsconfig.json file:
      1. npx tsc --init
      2. This file configures the TypeScript compiler options.
6. Integrating TypeScript with Existing JavaScript Projects
   1. Install TypeScript:
   2. npm install --save-dev typescript
7. Rename JavaScript files to TypeScript files:
   1. Change .js file extensions to .ts.
8. Run the TypeScript compiler:
   1. npx tsc
9. Basic Syntax and Types
   1. Basic Data Types
      1. number:
         1. let num: number = 5;
      2. string:
         1. let str: string = "Hello, TypeScript!";
      3. boolean:
         1. let isDone: boolean = true;
      4. null and undefined:
         1. let n: null = null;
         2. let u: undefined = undefined;
10. Understanding Type Annotations and Type Inference
    1. Type Annotations:
       1. let count: number = 10;
       2. let username: string = "John";
    2. Type Inference:
       1. let isActive = true; // Inferred as boolean
       2. let total = 42; // Inferred as number
    3. Static Typing
       1. Explanation of Static Typing and Its Benefits
       2. Static typing allows developers to define variable types, catching errors at compile-time and improving code quality and readability.
11. Interfaces

interface User {

name: string;

age: number;

}

1. Creating Interfaces for Object Shapes and Contracts

let user: User = {

name: "John",

age: 25

};

1. Optional Properties and Read-Only Properties in Interfaces
2. Optional Properties:

interface Car {

make: string;

model: string;

year?: number; // Optional property

}

1. Read-Only Properties:

interface Point {

readonly x: number;

readonly y: number;

}

1. Object-Oriented Programming Concepts in TypeScript
2. TypeScript supports classes and object-oriented programming principles such as encapsulation, inheritance, and polymorphism.
3. Defining Classes with Properties and Methods

class Person {

name: string;

age: number;

constructor(name: string, age: number) {

this.name = name;

this.age = age;

}

greet() {

return `Hello, my name is ${this.name}`;

}

}

1. Constructor: Initializes the object's properties.
2. Access Modifiers:
   1. public: Default, accessible from anywhere.
   2. private: Accessible only within the class.
   3. protected: Accessible within the class and subclasses.
3. Inheritance and Method Overriding

class Employee extends Person {

position: string;

constructor(name: string, age: number, position: string) {

super(name, age);

this.position = position;

}

greet() {

return `Hello, my name is ${this.name} and I am a ${this.position}`;

}

}

1. Generics

function identity<T>(arg: T): T {

return arg;

}

1. Creating Reusable Components with Generic Types

class Box<T> {

contents: T;

constructor(contents: T) {

this.contents = contents;

}

getContents(): T {

return this.contents;

}

}

1. Using Generic Constraints to Enforce Type Relationships

interface Lengthwise {

length: number;

}

function logLength<T extends Lengthwise>(arg: T): T {

console.log(arg.length);

return arg;

}

1. Advanced TypeScript Concepts
   1. Union Types and Intersection Types
      1. Union Types: A variable can hold one of several types.
         1. let id: number | string;
            1. id = 10; // Valid
            2. id = "ABC"; // Valid
   2. Intersection Types: Combines multiple types into one.
      1. interface A { a: number; }
      2. interface B { b: string; }
      3. type AB = A & B;
      4. let obj: AB = { a: 1, b: "hello" };
   3. Type Aliases and Type Assertions
      1. Type Aliases: Create new names for types.
         1. type StringOrNumber = string | number;
         2. let value: StringOrNumber;
            1. value = "hello";
            2. value = 123;
   4. Type Assertions: Override TypeScript's inferred type.
      1. let someValue: any = "this is a string";
      2. let strLength: number = (someValue as string).length;
   5. Type Guards for Working with Unions

function isNumber(value: string | number): value is number {

return typeof value === "number";

}

function logValue(value: string | number) {

if (isNumber(value)) {

console.log(`Number: ${value}`);

} else {

console.log(`String: ${value}`);

}

}

* 1. Understanding Conditional Types and Mapped Types
     1. Conditional Types: Perform type checks and return different types based on the result.
        1. type Check<T> = T extends string ? string : number;
  2. Mapped Types: Create new types by transforming properties of an existing type.

type Readonly<T> = {

readonly [P in keyof T]: T[P];

};