TypeScript Handbook

1. Introduction to TypeScript

Brief Overview of TypeScript

- TypeScript is a statically typed superset of JavaScript, meaning it extends JavaScript by adding static types. This allows developers to catch errors during development rather than at runtime.
- TypeScript code is transpiled into JavaScript, meaning it can run on any JavaScript runtime.

Advantages of TypeScript

- Provides type safety and error checking during development, reducing bugs and improving code quality.
- Enhances code maintainability and scalability by making code more self-documenting and easier to understand.
- Enables better tooling support and IDE features such as code navigation, refactoring, and intelligent code completion.

2. Getting Started

Installation Instructions

- Install TypeScript globally via npm: npm install -g typescript.
- This installs the TypeScript compiler (tsc) globally, allowing you to compile TypeScript files anywhere on your system.

Setting up a New Project

- Initialize a new TypeScript project using tsc --init. This creates a tsconfig.json file which specifies compiler options and project settings.
- Configure tsconfig.json according to your project requirements, such as specifying target ECMAScript version, output directory, and module system.

Integrating with Existing Projects

- Rename existing JavaScript files to TypeScript files (.js to .ts) to start using TypeScript.

- Begin adding type annotations gradually to existing JavaScript code, improving type safety and code quality over time.

3. Basic Syntax and Types

Overview of TypeScript Syntax

- TypeScript syntax is similar to JavaScript with the addition of static type annotations.
- Type annotations are added using the colon (:) syntax, for example: let count: number = 5;.

Basic Data Types

- TypeScript supports primitive data types such as number, string, boolean, null, and undefined.

Type Annotations and Inference

- Type annotations specify the type of a variable explicitly, while type inference allows TypeScript to infer types based on context.
  For example:
  let name: string = "John"; // Type annotation
  let age = 30; // Type inference, age is inferred as number

4. Static Typing

Explanation of Static Typing

- Static typing means that types are checked at compile time rather than runtime. This helps catch errors early in the development process.

Declaring Variable Types

- Variables can be explicitly typed using type annotations.
- For example:
  let count: number = 5;

Type Inference

- TypeScript can infer types based on the value assigned to a variable.
  For example:
  let message = "Hello, TypeScript!"; // message is inferred as string

5. Interfaces

Definition and Usage

- Interfaces define the shape of objects in TypeScript, specifying the names and types of properties.
- They are useful for describing the structure of objects that are expected to conform to a certain contract.
- Example:

```
interface Person {
  name: string;
  age: number;
}

function greet(person: Person) {
  console.log(`Hello, ${person.name}!`);
}
```

6. Classes

Object-Oriented Programming

- Classes in TypeScript allow developers to use object-oriented programming concepts such as encapsulation, inheritance, and polymorphism.

Constructors and Access Modifiers

- Constructors are special methods used for initializing class instances. Access modifiers (public, private, protected) control the visibility of class members.
- Example:

```
class Animal {
  private name: string;

  constructor(name: string) {
    this.name = name;
  }

  move(distance: number) {
```

```
        console.log(`${this.name} moved ${distance} meters.`);
    }
}
```

7. Generics

Introduction

- Generics allow developers to create reusable components that can work with a variety of data types.
- They provide a way to define functions and classes with placeholders for types.

Generic Constraints

- Generic types can be constrained to specific types or structures using type constraints. This ensures type safety and enables more precise type checking.
- Example:

```
function loggingIdentity<T extends Lengthwise>(arg: T): T {
    console.log(arg.length); // Error: Property 'length' does not exist on
type 'T'.
    return arg;
}
```

8. Advanced TypeScript Concepts

Union Types and Intersection Types

- Union types (|) allow a value to be one of several types, while intersection types (&) combine multiple types into one.

Type Aliases and Type Assertions

- Type aliases allow developers to create custom names for types, improving code readability.
- Type assertions (as) are a way to tell TypeScript that a value is of a specific type, overriding TypeScript's type inference.

Type Guards

- Type guards are runtime checks that help narrow down types within union types. They are useful for working with conditional logic.

Conditional Types and Mapped Types

- Conditional types allow developers to create types that depend on other types or conditions.
- Mapped types are used to dynamically transform existing types into new types based on certain rules or mappings.