Building Software Systems

Lecture 4.1

Introduction to TypeScript

SAURABH SRIVASTAVA

ASSISTANT PROFESSOR

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

IIT (ISM) DHANBAD

In this Unit

We will be having a look at Mobile App Development

- In particular, we will discuss the Ionic Framework for the same
- It requires a background in Angular and TypeScript
- We will start with TypeScript today
- In the next Lecture, we will have a look at Angular

What is TypeScript?

TypeScript is a typed superset of JavaScript that compiles to plain JavaScript

- Developed and maintained by Microsoft it is designed to develop large applications

Plain JavaScript has its own issues

- Plain JavaScript is *dynamically* typed – i.e., variables' type is assigned at the runtime
- This may lead to a number of problems

Dynamic Typing Problems in JavaScript

```javascript
// Function intended to add two numbers
function addNumbers(a, b) {
    return a + b;
}

// Expected use
console.log(addNumbers(5, 10)); // Output: 15

// Problematic use due to dynamic typing
console.log(addNumbers("5", "10")); // Output: "510" instead of the expected 15
```

What is TypeScript?

TypeScript is a typed superset of JavaScript that compiles to plain JavaScript

- Developed and maintained by Microsoft it is designed to develop large applications

Plain JavaScript has its own issues

- Plain JavaScript is *dynamically* typed – i.e., variables' type is assigned at the runtime
- This may lead to a number of problems

TypeScript intends to resolve these problems

- It adds static types to JavaScript variables
- It results in enhancing the development process by enabling error detection during compilation

TypeScript has multiple benefits

- Improves code quality and readability
- Facilitates early detection of potential errors
- Enhances editor support with autocompletion, navigation, and refactoring features

Type Annotations in TypeScript

Type annotations in TypeScript allow developers to explicitly declare the types

- Types can be defined for variables, function parameters, and return values
- Explicit type declaration helps the compiler understand the type of data at a particular point – at compile time
- Errors can be caught prior to testing, meaning problems in the code are caught beforehand
- These features can be helpful for Developer Tools and IDEs for better code hints

Basic Annotation syntax:

- Variable declarations: `let variableName: Type = value;`
- Function Params / Return Type:
  `function functionName(param1: Type, param2: Type): ReturnType { ... }`

Common Types in TypeScript:

- Primitive types: `string`, `number`, `boolean`, `null`, `undefined`
- Array type: `Type[]` or `Array<Type>`
- Object type notation: `{ propertyName: Type; anotherPropertyName: Type; }`

Basic Variable Declarations

```
let name: string = "Alice";
let age: number = 30;
let isStudent: boolean = false;
let address: null = null;
let jobTitle: undefined = undefined;
```

Array Declarations

```
// Using the 'Type[]' syntax
let numbers: number[] = [1, 2, 3, 4, 5];

// Using the 'Array<Type>' syntax
let fruits: Array<string> = ["Apple", "Banana", "Cherry"];
```

Object Declarations

```
let person: { name: string; age: number; isEmployed: boolean } = {
    name: "Alice",
    age: 30,
    isEmployed: true
};
```

Function Declarations

```
// Function intended to add two numbers with type annotations
function addNumbers(a: number, b: number): number {
    return a + b;
}

// Correct use
console.log(addNumbers(5, 10)); // Output: 15

// Incorrect use will cause TypeScript compilation error
// console.log(addNumbers("5", "10"));
// Error: Argument of type 'string' is not assignable to parameter of type 'number'.
```

Enums, Interfaces and Classes in TypeScript

Enums

- Enums (enumerations) allow you to define a set of named constants
- Plain JavaScript does not support Enums natively

Using Enums

typescript

```typescript
enum Direction {
  Up,
  Down,
  Left,
  Right
}
```

javascript

```javascript
const Direction = {
  Up: 0,
  Down: 1,
  Left: 2,
  Right: 3
};
// Usage: Direction.Up, Direction.Down, etc.
```

Enums, Interfaces and Classes in TypeScript

Enums

- Enums (enumerations) allow you to define a set of named constants
- Plain JavaScript does not support Enums natively

Interfaces

- Interfaces in TypeScript are used to define the structure of an object
- It specifies the expected form of an object without implementing any logic
- There is no real equivalent to an Interface in JavaScript
- Often, developers express the structure of an object in comments to do the same

Using Interfaces

typescript

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

javascript

```javascript
// There's no direct equivalent in JavaScript for TypeScript
// interfaces, but you might document the expected shape:
/**
 * Person object shape:
 * {
 *   name: string,
 *   age: number
 * }
 */
```

Enums, Interfaces and Classes in TypeScript

Enums

- Enums (enumerations) allow you to define a set of named constants
- Plain JavaScript does not support Enums natively

Interfaces

- Interfaces in TypeScript are used to define the structure of an object
- It specifies the expected form of an object without implementing any logic
- There is no real equivalent to an Interface in JavaScript
- Often, developers express the structure of an object in comments to do the same

Classes

- TypeScript and JavaScript both support classes from ECMAScript 2015 (ES6) onwards
- However, TypeScript classes can have additional features ….
- … such as access modifiers (public, private, protected), which are not available in JavaScript classes

Using Classes

typescript

```typescript
class Person {
  private name: string;
  constructor(name: string) {
    this.name = name;
  }
  greet() {
    console.log(`Hello, my name is ${this.name}`);
  }
}
```

javascript

```javascript
class Person {
  constructor(name) {
    this.name = name;
  }
  greet() {
    console.log(`Hello, my name is ${this.name}`);
  }
}
// Note: JavaScript does not support TypeScript's private keyword
// or type annotations.
```

Generics in TypeScript

Generics are a tool that enables the creation of components that can work with any data type

- They provide a way to ensure type safety without sacrificing flexibility

Generics is a feature commonly found in strongly typed languages like C++ or Java

- They are used in cases where the *same logic* may be applied to *multiple data types*
- Languages like JavaScript will not require you to declare types in such cases
- The problems that may be faced in these situations are those highlighted before (e.g., type mismatch)
- TypeScript provides generics as a feature

Common Use Cases:

- Function Example: `function identity<Type>(arg: Type): Type { return arg; }`
- Arrays: Array<Type> (e.g., `Array<number> = [1, 2, 3];`)

Functions with Generics

```
function identity<T>(arg: T): T {
    return arg;
}
let output = identity<string>("myString");  // Type of output is 'string'
```

Arrays with Generics

```
// A generic function to get the first element from an array of any type
function getFirstElement<T>(array: T[]): T | undefined {
    if (array.length === 0) return undefined;
    return array[0];
}

// Usage with number array
const numberArray: Array<number> = [10, 20, 30];
const firstNumber = getFirstElement(numberArray); // Type is 'number | undefined'

// Usage with string array
const stringArray: Array<string> = ["hello", "world"];
const firstString = getFirstElement(stringArray); // Type is 'string | undefined'

console.log(firstNumber); // Output: 10
console.log(firstString); // Output: "hello"
```

Linear Search with Generics

typescript

```typescript
function linearSearch<T>(array: T[], targetItem: T): number | null {
  for (let i = 0; i < array.length; i++) {
    if (array[i] === targetItem) {
      return i; // Return the index of the item if found
    }
  }
  return null; // Return null if the item is not found
}

// Example usage with number array
const numbers = [5, 3, 7, 1];
const index = linearSearch(numbers, 7); // Returns: 2

// Example usage with string array
const names = ["Alice", "Bob", "Charlie"];
const nameIndex = linearSearch(names, "Charlie"); // Returns: 2

console.log(`Index of number: ${index}`);
console.log(`Index of name: ${nameIndex}`);
```

javascript

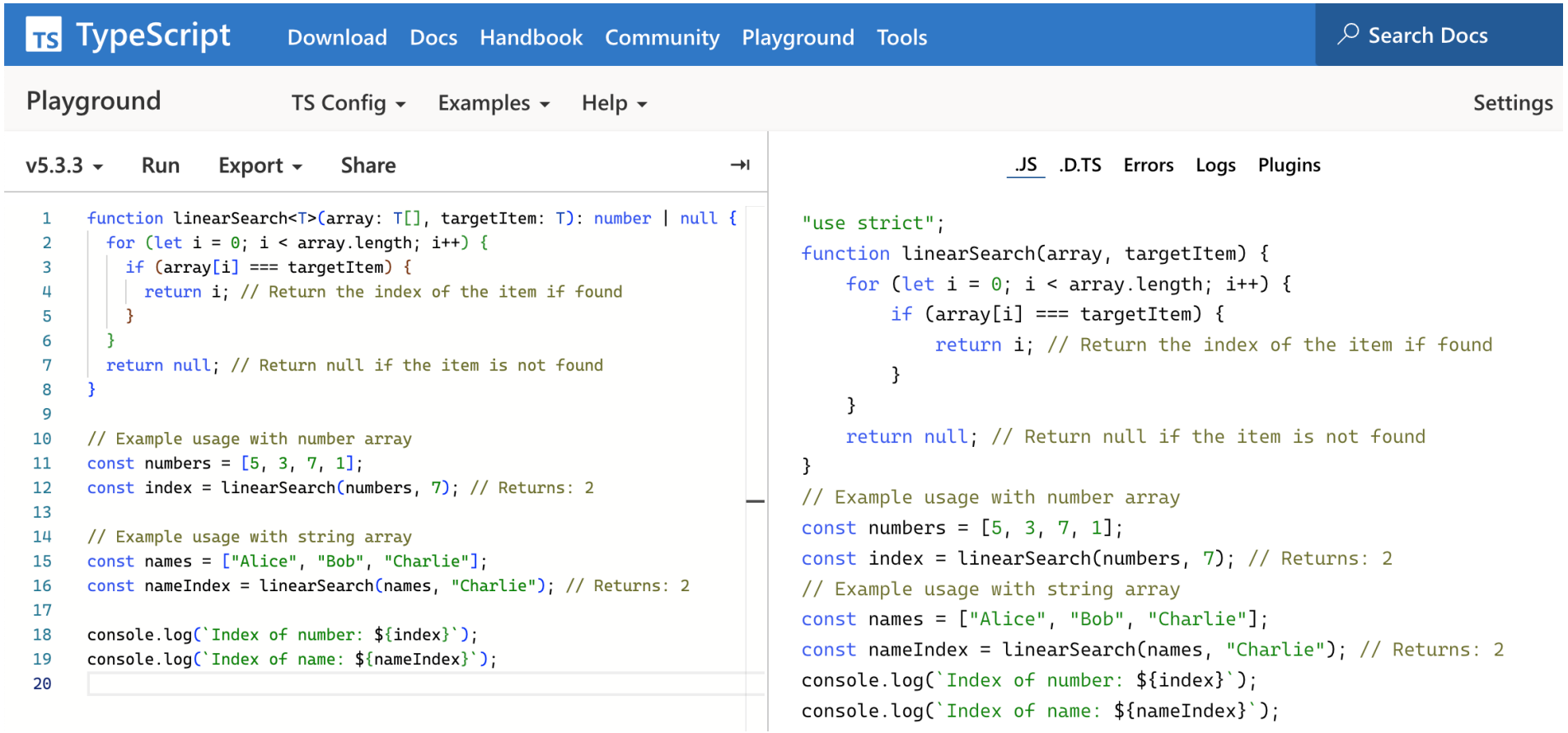
```javascript
function linearSearch(array, targetItem) {
  for (let i = 0; i < array.length; i++) {
    if (array[i] === targetItem) {
      return i; // Return the index of the item if found
    }
  }
  return null; // Return null if the item is not found
}

// Example usage with number array
const numbers = [5, 3, 7, 1];
const index = linearSearch(numbers, 7); // Returns: 2

// Example usage with string array
const names = ["Alice", "Bob", "Charlie"];
const nameIndex = linearSearch(names, "Charlie"); // Returns: 2

console.log(`Index of number: ${index}`);
console.log(`Index of name: ${nameIndex}`);
```

Trying out TypeScript in the TS Playground



Source: [Online TS Playground]

Homework

Go through the basic TypeScript Tutorial

- https://www.typescriptlang.org/docs/handbook/2/basic-types.html