Introduction to TypeScript

SoftUni Team

Technical Trainers

Software University

https://softuni.bg

Have a Question?

Software University

sli.do

#TypeScript

Table of Contents

1. Introduction to TypeScript
2. TypeScript vs JavaScript
3. Environment and Setup
4. Basic Data Types
5. Type Safety
6. Debugging

Introduction to TypeScript

What is TypeScript?

- TypeScript is an **open-source** programming language developed by Microsoft
- It is a **statically typed** superset of JavaScript that transpiles to plain JavaScript
- It uses **Static Analysis** that provides automated checking of your code **without actually running it**
- TypeScript adds optional **static typing**, making it more robust and maintainable

Why Use TypeScript?

- Static Typing: Helps catch errors during development, improving **code quality** and **reliability**
- Better Tooling: Enhanced code editor support with **intelligent auto-completion**, navigation, and refactoring

Why Use TypeScript?

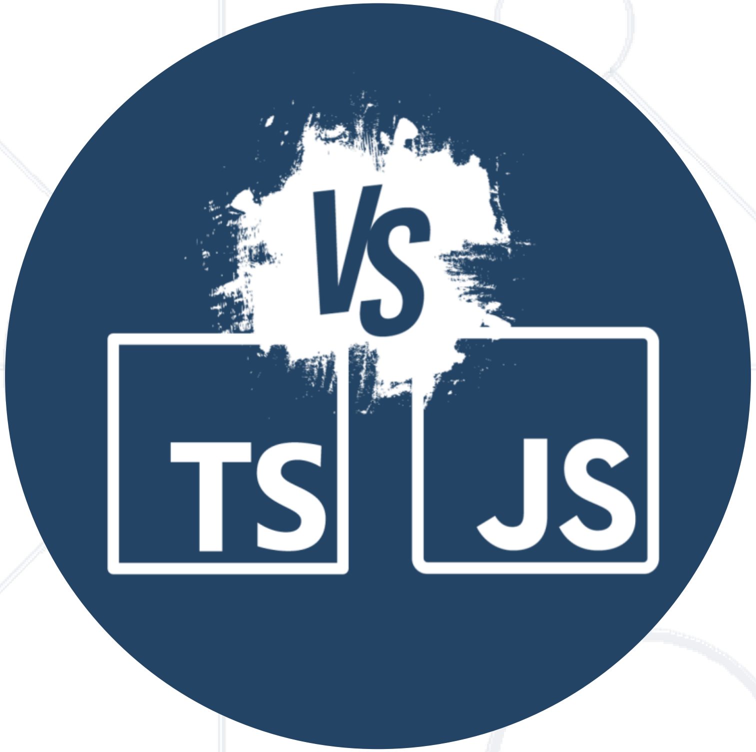
- Readability and Maintainability: Type annotations provide **self-documentation**, making code easier to understand and maintain
- Scalability: Suitable for **large-scale applications** with a strong type system

Key Features of TypeScript

- **Static Typing**: Types are **inferred** or **explicitly** declared, catching type-related errors during development
- **Interfaces**: Define contracts for **object shapes**, **enhancing code** readability and maintainability
- **Enums**: Define a set of **named constants** for improved code readability

Key Features of TypeScript

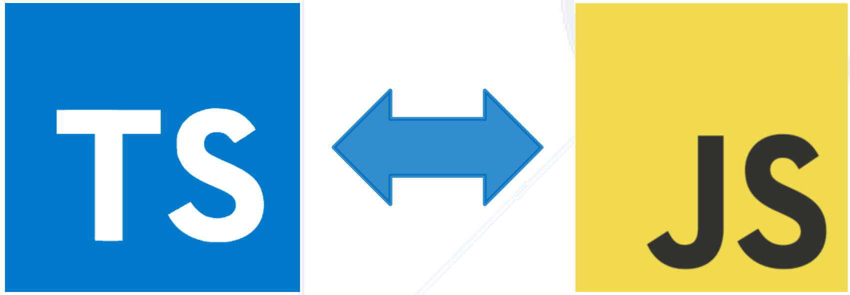
- Generics: Write **flexible** and **reusable** code components
- Decorators: **Extend** functionality or add **meta-data** to class members
- Improved OOP: Empowers Object Oriented Design, by introducing **interfaces**, **abstract classes** and **access modifiers**



TypeScript vs JavaScript

TypeScript vs JavaScript

- **JavaScript**: A **dynamic, loosely** typed language widely used for **web development**
- **TypeScript**: A **statically typed** superset of JavaScript that provides additional **features** and **tools** for better development experience



TypeScript vs JavaScript

- TypeScript

```
class Person {
  private firstName: string;
  constructor(fName: string) {
    this.firstName = fName;
  }
  greeting() {
    return `${this.firstName}`
  }
}
```

- JavaScript

```
"use strict";
class Person {
  constructor(fName) {
    this.firstName = fName;
  }
  greeting() {
    return  `${this.firstName}`;
  }
}
```

Environment and Setup

Install Visual Studio Code

- In this course we will use and demonstrate on:
  - Visual Studio Code
  - Installation Guidelines
- Alternatives:
  - WebStorm
  - JS Fiddle

Install TypeScript to Visual Studio Code

- Install **TypeScript** with **npm**

```
npm install -g typescript // latest stable build
```

- Test if **TypeScript** is **installed properly**

```
tsc --version // Should return a message 'Version 5.x.x'
```

- Create the **tsconfig.json** file

```
tsc --init // This command will create a new tsconfig.json file
```

Configuration of "tsconfig.json"

- In the tsconfig.json file, please **remove the comments** from the following:

```
{
  "compilerOptions" : {
      "target": "esnext",  // ECMAScript target version
      "module": "esnext",  // module code generation
      "sourceMap": true,   // Generates corresponding .map file
      "strict": true,      // strict type-checking options
      "outDir": "out",     // redirect output to the directory.
    }
}
```

Transpilation vs Compilation

- Transpilation
  - Source code is **translated** to a similar-level language.
  - Output is in a **similar abstraction** level
  - **Example:** TypeScript to JavaScript
- Compilation
  - Source code is translated to a **lower-level language**
  - Output is in a form suitable for **direct execution** by the machine

Basic Data Types

Basic Data Types

- **String** - used to represent **textual** data

```
let str: string = 'hello';
str = 'singleQuotes' ; // valid
str = "doubleQuotes" ; // valid
str = 11; // invalid
```

- **Number** - used to represent **numeric** data

```
let decimal: number = 11; // valid
let hex: number = 7E3; // valid
let binary: number = 11111100011 // valid
let float: number = 3.14 // valid
decimal = 'hello'; // invalid
```

Basic Data Types

- **Boolean** - only **true** and **false** values
  - Functions or expressions that return **true** or **false** values may also be assigned to **Boolean** data type

```
let isBool: boolean = true;
isBool = 5 < 2; // valid
let numbers = [1, 2, 3, 4];
isBool = numbers.includes(100) // valid
isBool = 11; // invalid
```

Basic Data Types

- Symbol - used to represent **unique** data

```
let uniqueSymbol: symbol = Symbol('mySymbol');
let anotherSymbol: symbol = Symbol('mySymbol');
console.log(uniqueSymbol === anotherSymbol); // false
```

- null and undefined - represent the **absence** of a value in variables and functions

```
let undefinedValue1; // undefined
let undefinedValue2: undefined = undefined;
let person: null = null;
```

Basic Data Types

- **Array** - use any valid data type (String, Boolean, Number) and postfix []

```
let arrayOfStr: string[];
arrayOfStr.push('Hello'); // valid
arrayOfStr.push(11); // invalid
```

- **Tuple** - array with fixed number of elements whose types are known

```
let tuple:[string, number];
tuple = ['Hello', 11]; // valid
tuple = [11, 'Hello']; // invalid
```

Basic Data Types

- **Enum** - gives sets of numeric or string values more readable names
  - By default, each enum starts at 0

```
enum DaysOfTheWeek {
    Monday,  // 0
    Tuesday, // 1
    …
};
let day: DaysOfTheWeek;
day = DaysOfTheWeek.Monday;
console.log(day); // 0
if (day === DaysOfTheWeek.Monday) {
    console.log('I hope you all had a great weekend!');
} // It will print the message
```

Basic Data Types

- **Any** - takes any value and skips all type checks
- **Unknown** - takes any value, but type checks are still done, useful since it forces type narrowing/assertions

```
let a: any = 'hello'; let b: unknown = 'hello';
a = 11; b = 12;
console.log(a.length); // allowed, skips all type checks
console.log(b.length); // TS Error: Property 'length'
does not exist on type 'unknown'
```

- **Void** - used in functions that return no value

```
function greet(message: string): void {
    console.log(message);
}
```

Optional Data Types

- The **optional** data types are marked with **?**
  - Required parameters **cannot** follow optional ones

```
function optionalParams(name: string, mail?: string) {
      // some logic
} // valid

function optionalParams(name?: string, mail: string) {
      // some logic
} // invalid
```

Return Data Types

- The **return data types** are marked with **:** after the braces in function declaration
  - The **return value type** should match the **return type**

```
function greet (name: string): string {
    return name;
}

console.log(greet('Hello'));
```

Type Safety

Type Inference

- Type inference allows TypeScript to **automatically deduce types**, **improving code readability** and **development speed**

```
// here the type is automatically inferred to
// { code: number, text: string}
let httpCode = {
 code: 404,
 text: 'Page not found'
};
```

Type Assertions

- Allow you to pass type information to Typescript
- **Does not actually change** the underlying value
- Can be done using **<>** or the **as** keyword

```
let val:unknown = 20;
let str = val as string;

//no TypeScript error
console.log(str.length);                // undefined
console.log((<string>val).length);  // undefined

//TS error, as it expects 'str' to be a string
console.log(str * 10);  // 200
console.log(typeof str)     // number
```

Type Guards

- Any expression that allows TypeScript to **narrow the type** information **in some scope**, like **typeof**, **type predicate function**, **instanceof** and more

```
function createRandomVariable(): unknown {…}
// type predicate function
function isString(val: unknown): val is string {
    // TS allows charAt call, since we assert val is a string
    return (val as string).charAt != undefined);
}

let myVal: unknown = createRandomVariable();
console.log(myVal.length);       // Error
console.log(myVal * 2);          // Error
if(isString(myVal)) console.log(myVal.length);          // valid
if(typeof myVal === 'string') console.log(myVal.length) // valid
```

Debugging

Debugging in VS Code

- Utilizing VS Code's powerful **integrated debugger** to find and fix issues in your TypeScript code
- Setting **breakpoints**, inspecting **variables**, and stepping through code

Debugging in VS Code

- Initialize a **TypeScript Project**:
  - Create a **tsconfig.json** file to configure TypeScript settings for the project
- **Launch Configurations**:
  - Configure a **launch.json** file to define how VS Code launches the debugging process
  - Set up configurations for **different scenarios**

Configuration of "launch.json"

- Choose the **create a launch.json file** option from the Debug tab
- Replace the contents of **launch.json** with the following:

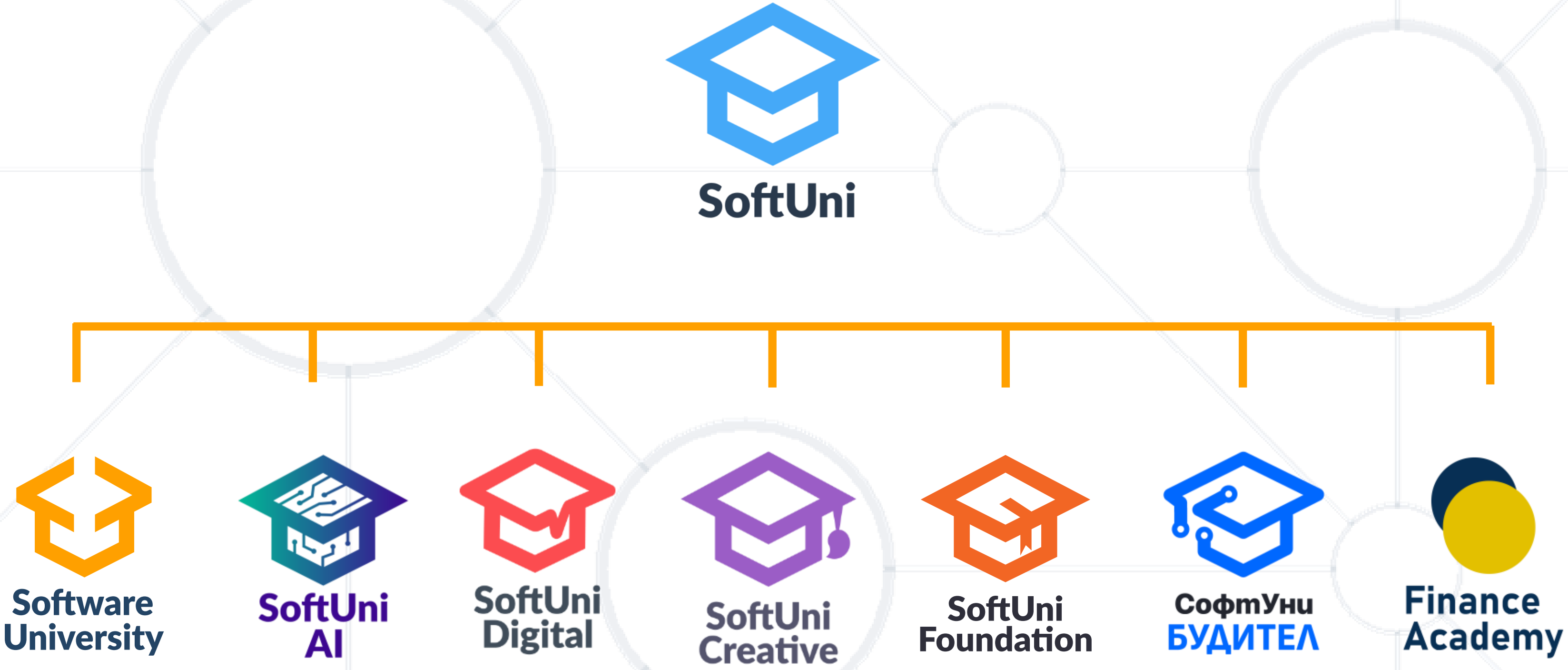
```
{
    "version": "0.2.0",
    "configurations": [
        {
            "type": "node",
            "request": "launch",
            "name": "Launch Program",
            "program": "${workspaceFolder}/${fileBasename}",  // Run the currently opened file
            "preLaunchTask": "tsc: build - tsconfig.json",  // Transpile the files
            "outFiles": [
                "${workspaceFolder}/out/**/*.js" // Look for the transpiled files in /out dir
            ]
        }
    ]
}
```

Summary

- TypeScript presents strong typing to your JavaScript code
  - let, const and var are used to declare variables
  - You can use basic types (Number, String, Boolean, Enum, etc.)
  - You can use type guards and type assertions to specify type information
- Functions can:
  - Take optional and required parameters and return a result

Software University

Questions?


SoftUni

Software University

SoftUni AI

SoftUni Digital

SoftUni Creative

SoftUni Foundation

СофтУни БУДИТЕЛ

Finance Academy

© SoftUni – https://about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

SoftUni Diamond Partners

Coca-Cola HBC Bulgaria

Postbank

top EMPLOYER България Bulgaria 2025

FOR A BETTER WORLD OF WORK

SUPER HOSTING .BG

createX

DRAFT KINGS

THE CROWN IS YOURS

INDEAVR

Serving the high achievers

encorp.io

VIVACOM

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg/
- © Software University – https://softuni.bg