Introduction to TypeScript

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://about.softuni.bg

Have a Question?

Software University

sli.do

#typescript

Table of Contents

Introduction to TypeScript

What is TypeScript?

Software University

- TypeScript is an **open-source** programming language developed by Microsoft
- It is a **statically typed** superset of JavaScript that transpires to plain JavaScript
- TypeScript adds optional **static typing**, making it more robust and maintainable

Why Use TypeScript?

Software University

- **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? (2)

Software University

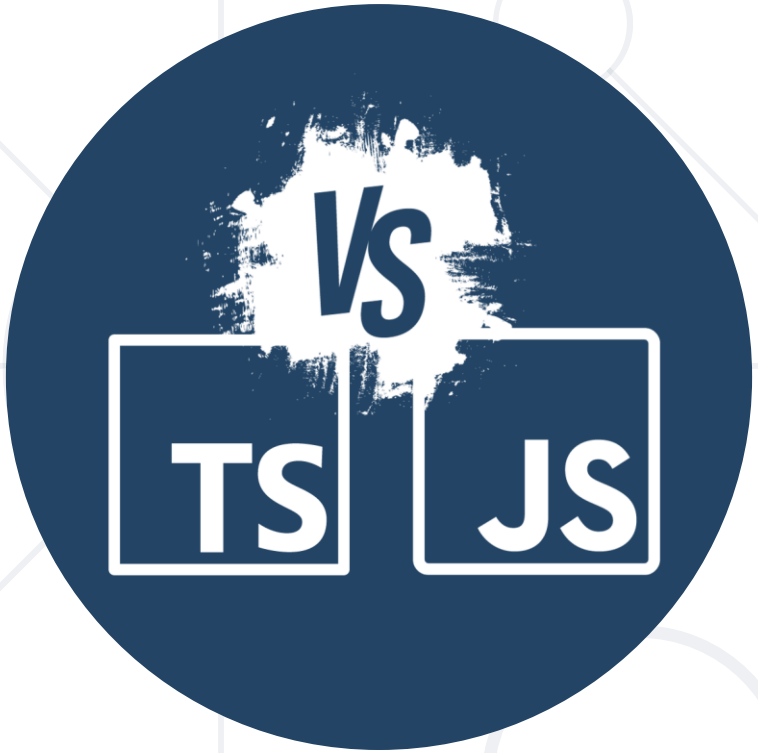
- **Readability and Maintainability**: Type annotations provide self-documentation, making code easier to understand and maintain
- **Scalability**: Suitable for large-scale applications with a strong type system

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
- **Classes**: Follow object-oriented principles with support for constructors, properties, and methods

Key Features of TypeScript (2)

- **Enums**: Define a set of named constants for improved code readability
- **Generics**: Write flexible and reusable code components
- **Modules**: Organize code into logical and reusable units for better maintainability

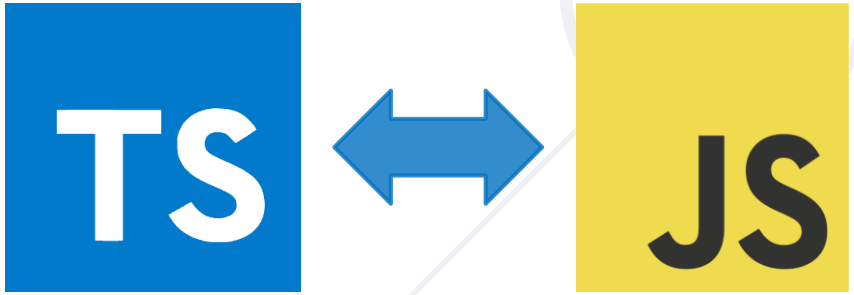


TypeScript vs JavaScript

TypeScript vs JavaScript

Software University

- **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

Software University

- 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"

Software University

- 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 (e.g., machine code)
  - Output is in a form suitable for direct execution by the machine

Basic Data Types

Basic Data Types

Software University

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

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

- **Number** - a numeric data type

```
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

Software University

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

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

- **null and undefined** - special types used to represent 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('World'); //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 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** and **Unknown** - takes any and all values. It's a way to escape the strong types. Unknown is safer.

```
let a: any = 'hello'; // let a: unknown = 'hello';
a = true; //valid
a = 11 ; //valid
```

- **Void** - mainly 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'));
```

Advanced Data Types

Advanced Data Types (1)

Software University

- **Union type** - combine multiple types in one type

```
function greet(message: string | string[]) {
    if (typeof message === "string") {
        return message;
    }
    return message.join(' ');
}
let greeting = 'Hello world';
let greetingArray = ['Dear', 'Sir/Madam'];

console.log(greet(greetingArray)); //Dear Sir/Madam
```

Advanced Data Types (2)

- **Intersection types** - combine multiple types in one type

```
interface Person { fullName: string | string[]; }
interface Contact { email: string; }
function showContact(contactPerson: Person & Contact) {
    return contactPerson;
}
let contactPerson: Person & Contact = {
    fullName: 'Svetoslav Dimitrov',
    email: 'test@test.com'
}
console.log(showContact(contactPerson));
```

Literal Types

Software University

- String Literal Type

```
let status: "success" | "error";
status = "success"; // valid
```

- Number Literal Type

```
let errorCode: 200 | 400 | 404;
errorCode = 200; // valid
```

Type Aliases

Software University

- **Simple Type Alias**

```
type Age = number;
const myAge: Age = 25;
```

- **Object Type Alias**

```
type User = { id: number; name: string; };
const user: User = { id: 1, name: 'John Doe' };
```

"keyof" usage:

- Retrieves the keys of an object type as a union of string or numeric literals

```
type Point = { x: number; y: number; };
type PointKeys = keyof Point; // 'x' | 'y'

type Colors = { red: string; blue: string; };
type ColorKeys = keyof Colors; // 'red' | 'blue'
```

Mapped Types:

Software University

- Creates new types by transforming each property of an existing type.

```
type Optional<T> = { [K in keyof T]?: T[K] };
type PartialPoint = Optional<Point>;
// { x?: number; y?: number; }

type Readonly<T> = { readonly [K in keyof T]: T[K] };
type ReadonlyColors = Readonly<Colors>;
// { readonly red: string; readonly blue: string; }
```

Recursive types and Interfaces

Software University

- **Recursive types** are vital for representing complex, self-referential data structures
- **Type inference** allows TypeScript to automatically deduce types, improving code readability and development speed

```
interface TreeNode {
 value: number;
 left?: TreeNode;
 right?: TreeNode;
}
```

Debugging

Debugging in VS Code (1)

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

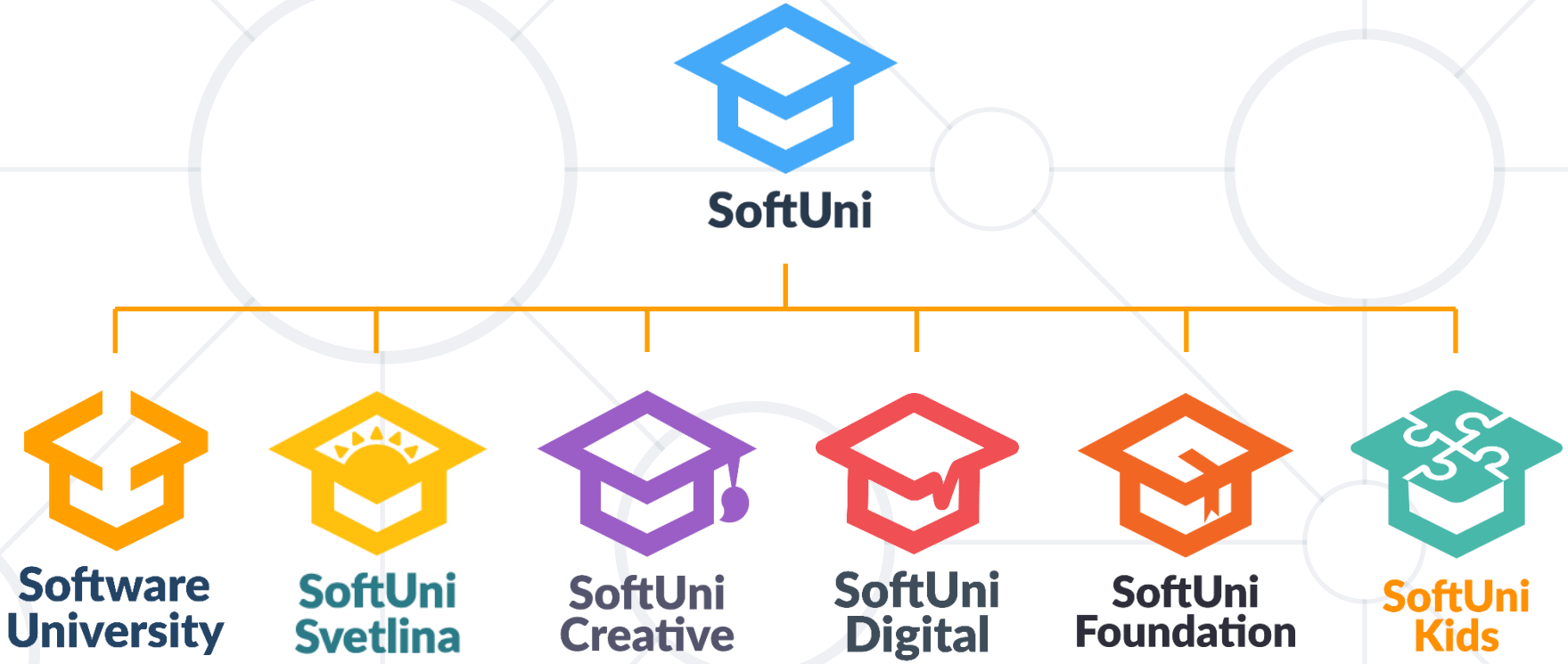
Debugging in VS Code (2)

- 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 (e.g., debugging a file, a Node.js application, etc.).

Summary

- TypeScript presents strong typing to your JavaScript code
  - `let`, `const` and `var` are used to declare variables
  - There are basic (Number, String, Boolean, etc.) and more advanced data types like union or intersection
- Functions can:
  - Take optional and required parameters and return result

Questions?



SoftUni Diamond Partners

Software University

SUPER HOSTING .BG

Coca-Cola HBC Bulgaria

Flutter™ International

INDEAVR
Serving the high achievers

AMBITIONED

DRAFT KINGS

SOFTWARE GROUP

BOSCH

Postbank
Решения за твоето утре

PHAR VISION

SmartIT

DXC TECHNOLOGY

createX

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
- Software University Forums
  - forum.softuni.bg

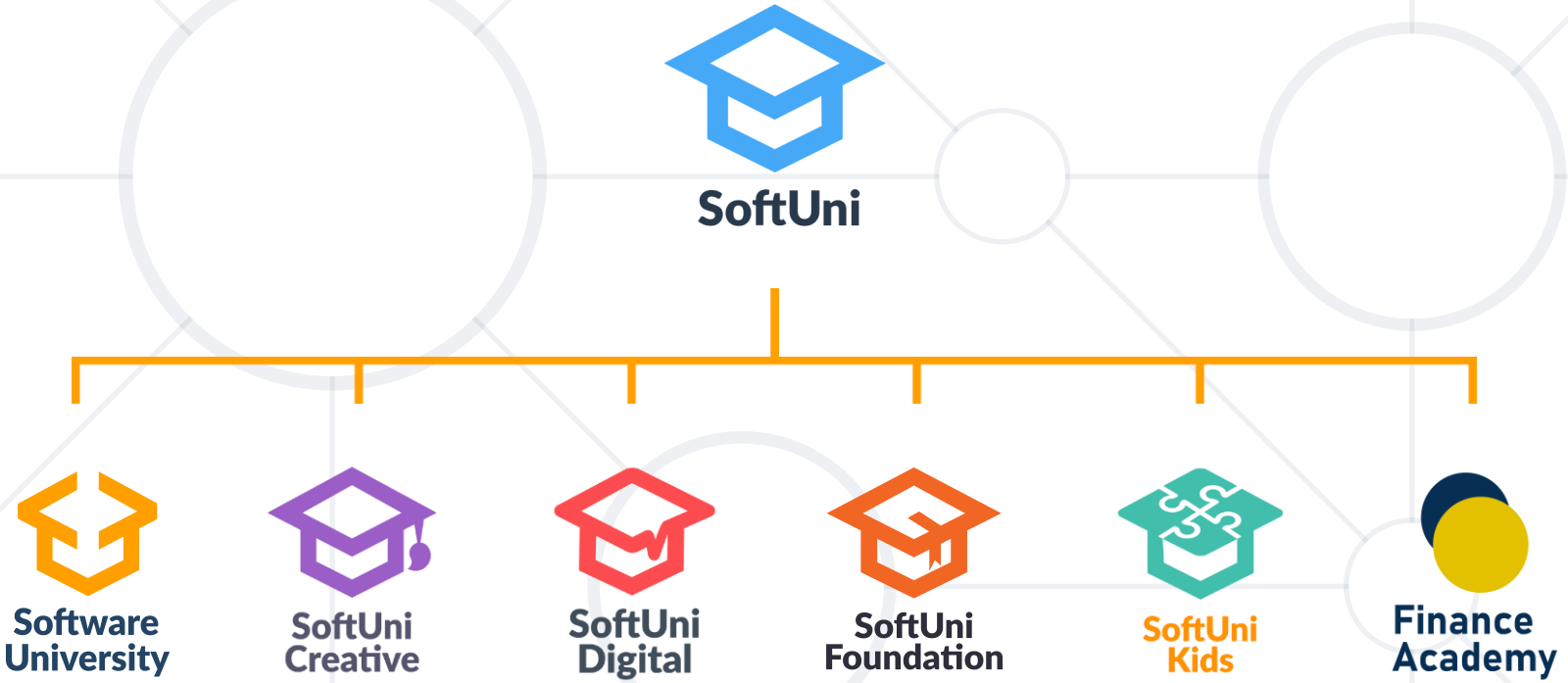
License

- 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

Summary

- TypeScript presents strong typing to your JavaScript code
  - `let`, `const` and `var` are used to declare variables
  - There are basic (Number, String, Boolean, etc.) and more advanced data types like union or intersection
- Functions can:
  - Take optional and required parameters and return result

Questions?



SoftUni Diamond Partners

Software University

SUPER HOSTING .BG

Coca-Cola HBC Bulgaria

Flutter™ International

INDEAVR
Serving the high achievers

AMBITIONED

DRAFT KINGS

SOFTWARE GROUP

BOSCH

Postbank
Решения за твоето утре

PHAR VISION

SmartIT

DXC TECHNOLOGY

createX

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
- Software University Forums
  - forum.softuni.bg

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