Typescript Development

“Superset of JavaScript that compiles into JS”

DotNetTricks
CODING IS RHYME

About Me

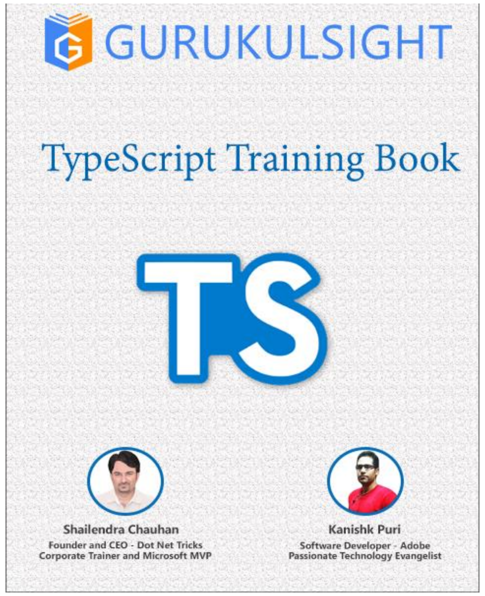
Hi, I’m Shailendra Chauhan

- Author
- Architect,
- Corporate Trainer
- Microsoft MVP
- Founder and CEO of Dot Net Tricks

@proshailendra

@proshailendrachauhan

Author of Most Popular Free e-Books

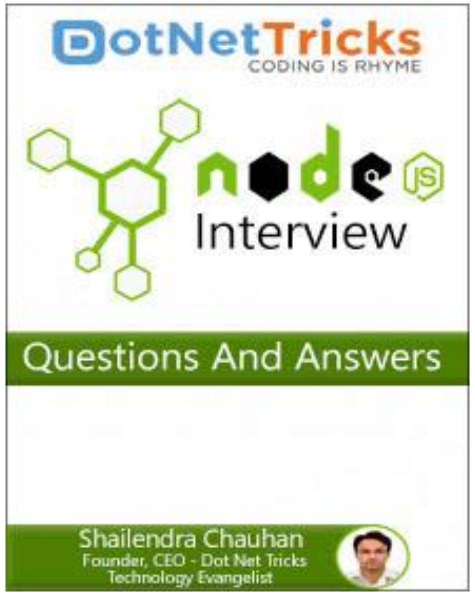

GURUKULSIGHT

TypeScript Training Book

TS

Shailendra Chauhan
Founder and CEO - Dot Net Tricks
Corporate Trainer and Microsoft MVP

Kanishk Puri
Software Developer - Adobe
Passionate Technology Evangelist


DotNetTricks
CODING IS RHYME

node JS Interview

Questions And Answers

Shailendra Chauhan
Founder, CEO - Dot Net Tricks
Technology Evangelist

DotNetTricks
CODING IS RHYME

ANGULARJS 1.x Interview

Questions And Answers

Shailendra Chauhan
Founder, CEO - Dot Net Tricks
Technology Evangelist

DotNetTricks
CODING IS RHYME

ASP.NET MVC Interview

Questions And Answers

Shailendra Chauhan
Founder, CEO - Dot Net Tricks
Technology Evangelist

DotNetTricks
CODING IS RHYME

LINQ Interview

Questions And Answers

Shailendra Chauhan
Founder, CEO - Dot Net Tricks
Technology Evangelist

Agenda

- Introduction to TypeScript
- Issues with Plain JavaScript
- Advantages of TypeScript
- Getting Started with TypeScript
- Data Types
- Functions
- Classes
- Access Modifiers
- Inheritance

Agenda

- Method Overloading
- Interfaces
- Generics
- Modules
- Namespaces

Introduction to TypeScript

DotNetTricks
CODING IS RHYME

Introduction to TypeScript

- Super set of JavaScript that compiles into JavaScript
- Developed by Microsoft and released 1.0 version in 2012
- Free and open source programming language
- Supports type-safety, data types, classes, interfaces, inheritance, modules and much more..
- Supports latest standard and evolving features of JS including ES5 and ES6

Issues with Plain JavaScript

Issues with Plain JavaScript

- Dynamic Typing
- Lack of Type Safety
- Lack of OOPs concept like classes, Interfaces
- Lack of modularity

Advantages of TypeScript

Advantages of TypeScript

- Simplify code which is easier to read and debug
- Provides Type Safety at compile time
- It’s compiled, rather than interpreted
- Unlike JavaScript, it uses both interfaces & classes
- Supports access modifiers and modularity
- Easy to adopt for object oriented developers (like java & C#)
- Open source

Getting Started with TypeScript

Getting Started with TypeScript

- Development IDEs : Sublime Text, WebStrom, Eclipse, Visual Studio, Visual Studio Code etc.
- Supports Any browser, Any host and Any OS

Visual Studio 2017	Sublime Text	Emacs
Visual Studio 2015	Atom	WebStorm
Visual Studio Code	Eclipse	Vim

DotNetTricks
CODING IS RHYME

Data Types

Data Types

- Any
- Built-In

  number, string, boolean, void, null, undefined
- User Defined

  class, interface, enum, array, function

Functions

DotNetTricks
CODING IS RHYME

TS

Functions

- Named Function
- Anonymous Function/ Function Expression
- Arrow Function

Functions

function.ts

```
//named function with number as parameters type and return type
function add(x: number, y: number): number {
    return x + y;
}
//anonymous function with number as parameters type and return type
let sum = function (x: number, y: number): number {
    return x + y;
};
```

arrowfunction.ts

```
//arrow function with typed parameters
let add = (x: number, y: number)=> {
    return x + y;
};

let result = add(2, 3); //5
```

Classes

Classes

- Instance members/methods
- Constructor – Default, Parameterized
- Default/optional Parameters
- ES6 class syntax
- Static members/methods
- Inheritance
- Implements interfaces

Classes

class.ts

```
class Student {
    private rollNo: number;
    private name: string;

    constructor(_rollNo: number, _name: string) {
        this.rollNo = _rollNo;
        this.name = _name;
    }
    showDetails() { //public : by default
        console.log(this.rollNo + " : " +
this.name);
    }
}

let s1 = new Student(1, "Shailendra Chauhan");
s1.showDetails(); //1 : Shailendra Chauhan

let s2 = new Student(2, "kanishk Puri");
s2.showDetails(); //2 : kanishk Puri
```

class.js

```
var Student = (function () {
    function Student(_rollNo, _name) {
        this.rollNo = _rollNo;
        this.name = _name;
    }
    Student.prototype.showDetails = function () {
        console.log(this.rollNo + " : " + this.name);
    };
    return Student;
}());
var s1 = new Student(1, "Shailendra Chauhan");
s1.showDetails();

var s2 = new Student(2, "kanishk Puri");
s2.showDetails();
```

Access Modifiers

Access Modifiers

- Public (by default)
- Private
- Protected

Classes

class.ts

```
class Student {
    private rollNo: number;
    private name: string;

    constructor(_rollNo: number, _name: string) {
        this.rollNo = _rollNo;
        this.name = _name;
    }
    showDetails() { //public : by default
        console.log(this.rollNo + " : " +
this.name);
    }
}

let s1 = new Student(1, "Shailendra Chauhan");
s1.showDetails(); //1 : Shailendra Chauhan

let s2 = new Student(2, "kanishk Puri");
s2.showDetails(); //2 : kanishk Puri
```

class.js

```
var Student = (function () {
    function Student(_rollNo, _name) {
        this.rollNo = _rollNo;
        this.name = _name;
    }
    Student.prototype.showDetails = function () {
        console.log(this.rollNo + " : " + this.name);
    };
    return Student;
}());
var s1 = new Student(1, "Shailendra Chauhan");
s1.showDetails();

var s2 = new Student(2, "kanishk Puri");
s2.showDetails();
```

DotNetTricks
CODING IS RHYME

Constructors

DotNetTricks
CODING IS RHYME

Constructors

- Supports two types of constructors - default and parameterized
- Supports Constructor Overloading
- Unlike C#, In the constructor, you can make public or private instance members of a class

constructor.ts

```
class Customer {
 //instance members with access modifiers
 constructor(private id:number, public name:string, protected address:string) { }
    showDetails() {
        console.log(this.id + " : " + this.name + " : " + this.address);
    }
}
let c1 = new Customer(1, "Shailendra Chauhan", "Noida");
c1.showDetails(); //1 : Shailendra Chauhan : Noida
```

Inheritance

Inheritance

- Single Level
- Multi Level

inheritance.ts

```
class Person {
    private firstName: string;
    private lastName: string;
    constructor(_firstName: string, _lastName: string) {
        this.firstName = _firstName;
        this.lastName = _lastName;
    }
    fullName(): string {
        return this.firstName + " " + this.lastName;
    }
}
class Employee extends Person {
    id: number;
    constructor(_id: number, _firstName: string, _lastName: string) {
        //calling parent class constructor
        super(_firstName, _lastName);
        this.id = _id;
    }
    showDetails(): void {
        console.log(this.id + " : " + this.fullName()); //calling parent class method
    }
}
let e1 = new Employee(1, "Shailendra", "Chauhan");
e1.showDetails(); //1 : Shailendra Chauhan
```

Function Overloading

FunctionOverloading

- Based on numbers of parameters only

functionoverloads.ts

```
function add(x: string, y: string, z: string): string;
function add(x: number, y: number, z: number): number;

// implementation signature
function add(x: any, y: any, z: any): any {
    let result: any;
    if (typeof x == "number" && typeof y == "number" && typeof z == "number") {
        result = x + y + z;
    }
    else {
        result = x + y + " " + z;
    }
    return result;
}

let result1 = add(4, 3, 8); // 15
let result2 = add("Gurukul", "sight", "website"); //Gurukulsight website
```

Interfaces

Interfaces

- Acts as a contract between itself and any class which implements it
- A class that implement an interface is bound to implement all its members
- Interface cannot be instantiated but it can be referenced by the class object which implements it.
- Interfaces can be used to represent any non-primitive JavaScript object.

Interfaces

interface.ts

```
interface IHuman {
    firstName: string;
    lastName: string;
}
class Employee implements IHuman {
    constructor(public firstName: string, public lastName: string) {

    }
}
```

Generics

Generics

- Enforce type safety without compromising performance, or productivity
- A type parameter is supplied between the open (<) and close (>) brackets which makes it to allow similar types of objects
- TypeScript supports generic functions, generic interfaces and generic classes

Generics

genericfunction.ts

```
function doReverse<T>(list: T[]): T[] {
    let revList: T[] = [];
    for (let i = (list.length - 1); i >= 0; i--) {
        revList.push(list[i]);
    }
    return revList;
}
let letters = ['a', 'b', 'c', 'd', 'e'];
let reversedLetters = doReverse<string>(letters); // e, d, c, b, a

let numbers = [1, 2, 3, 4, 5];
let reversedNumbers = doReverse<number>(numbers); // 5, 4, 3, 2, 1
```

Generics

genericclass.ts

```
class ItemList<T>
{
    private itemArray: Array<T>;
    constructor() {
        this.itemArray = [];
    }
    Add(item: T) : void  {
        this.itemArray.push(item);
    }
    GetAll(): Array<T> {
        return this.itemArray;
    }
}
let fruits = new ItemList<string>();
fruits.Add("Apple");
fruits.Add("Mango");
fruits.Add("Orange");

let listOfFruits = fruits.GetAll();
for (let i = 0; i < listOfFruits.length; i++) {
    console.log(listOfFruits[i]);
}
```

Modules

Modules

- Acts as a container to a group of related variables, functions, classes, and interfaces etc.
- Use export keyword to access Variables, functions, classes, and interfaces etc. declared in a module outside the module
- Use import keyword to consume the members of a module

Modules

myModule.ts

```
//exporting Employee type
export class Employee {
    constructor(private firstName: string, private lastName:
string) { }
    showDetails() {
        return this.firstName + ", " + this.lastName;
    }
}

//exporting Student type
export class Student {
    constructor(private rollNo: number, private name:
string) { }
    showDetails() {
        return this.rollNo + ", " + this.name;
    }
}
```

app.ts

```
//importing the exporting types Student and
Employee from myModule file
import { Student, Employee } from "./myModule";

let st = new Student(1, "Mohan");
let result1 = st.showDetails();
console.log("Student Details :" + result1);

let emp = new Employee("Shailendra", "Chauhan");
let result2 = emp.showDetails();
console.log("Employee Details :" + result2);
```