**Type Script**

TypeScript is an open-source object-oriented language developed and maintained by Microsoft, It is a typed superset of Javascript that compiles to plain JavaScript.

TypeScript was developed under Anders Hejlsberg, who also led the creation of the C# language.

TypeScript was first released in October 2012.

**Prerequisites**

Basic knowledge of JavaScript is required.

TypeScript extends JavaScript by adding **data types, classes**, and other

**object-oriented features** with **type-checking**. It is a typed superset of

JavaScript that compiles to plain JavaScript.

**TypeScript Version History**

Version Released Date

* TypeScript 0.8 October 2012
* TypeScript 0.9 June 2013
* TypeScript 1.0 October 2014
* TypeScript 2.0 September 2016
* TypeScript 3.0 July 2018
* TypeScript 4.0 August 2020
* TypeScript 4.4.0 -sep 2021
* TypeScript 4.5 17 November 2021
* TypeScript 4.6 28 February 2022
* TypeScript 4.7 24 May 2022
* TypeScript 4.8 25 August 2022

The type system increases the code quality, readability and makes it easy to

maintain and refactor code base. More importantly, errors can be caught at

compile time rather than at runtime.

**TypeScript Features**

**Cross-Platform:** TypeScript runs on any platform that JavaScript runs on.

TypeScript compiler can be installed on any Operating System such as

Windows, macOS, and Linux.

**Object-Oriented Language:**

TypeScript provides powerful features such as Classes, Interfaces, and

Modules. You can write pure object-oriented code for client-side as well as

server-side development.

**Static type-checking:**

TypeScript uses static typing. This is done using type annotations. It helps

type checking at compile time. Thus, you can find errors while typing the code without running your script each time. Additionally, using the type inference mechanism, if a variable is declared without a type, it will be inferred based on its value.

**Optional Static Typing:** TypeScript static typing is optional, if you prefer to use JavaScript's dynamic typing.

**DOM Manipulation:** Like JavaScript, TypeScript can be used to manipulate the DOM.

(European Computer Manufacturer's Association)

**ES 6 Features:** TypeScript includes most features of planned ECMAScript

2015 (ES 6, 7) such as class, interface, Arrow functions etc.

**TypeScript Advantages**

* TypeScript is an open-source language with continuous development and maintenance by Microsoft.
* TypeScript runs on any browser or JavaScript engine.
* TypeScript is similar to JavaScript and uses the same syntax and
* semantics.
* All of TypeScript's code finally gets converted into JavaScript.
* This allows a quicker learning curve for front-end developers currently

coding in JavaScript.

* TypeScript is also closer in syntax to backend languages like Java and
* Scala. This helps backend developers write front-end code faster.
* TypeScript code can be called from an existing JavaScript code.
* TypeScript also works with existing JavaScript frameworks and libraries
* without any issues.
* The TypeScript Definition file, with .ts extension, provides support for
* existing JavaScript libraries like Jquery, D3.js, etc. So,
* TypeScript code can add JavaScript libraries using type definitions to avail the benefits of type-checking, code autocompletion, and documentation in existing dynamically-typed JavaScript libraries.
* TypeScript has support for the latest JavaScript features from
* ECMAScript 2015 . It includes features from ES6 and ES7 that can run in ES5-level JavaScript engines like Node.js.

**install**

install node.js

npm install -g typescript

tsc -v

**working with class**

classes are the fundamental entities which are used to create reusable

components.

A class definition can include the following −

**Fields** − A field is any variable declared in a class. Fields represent data

pertaining to objects

**Constructors** − Responsible for allocating memory for the objects of the

class

**Functions** − Functions represent actions an object can take. They are also at

times referred to as methods

Syntax to declare a class

A class keyword is used to declare a class in TypeScript. We can create a

class with the following syntax:

class <class\_name>{

field;

method;

}

**CReating an object of class**

A class creates an object by using the new keyword followed by the class

Name The new keyword allocates memory for object creation at runtime. All objects get memory in heap memory area.

let object\_name = new class\_name(parameter)

class Employee {

empCode: number;

empName: string;

constructor(code: number, name: string) {

this.empName = name;

this.empCode = code;

}

getSalary() : number {

return 10000;

}

}

class Car {

//field

engine:string;

//constructor

constructor(engine:string) {

this.engine = engine

} //

function

disp():void {

console.log("Function displays Engine is : "+this.engine)

}

} //

create an object

var obj = new Car("abctest")

//access the field

console.log("Reading attribute value Engine as : "+obj.engine)

//access the function

obj.disp()

**Class Inheritance**

Inheritance is the ability of a program to create new classes from an existing

class.

The class that is extended to create newer classes is called the parent

class/super class. The newly created classes are called the child/sub classes.

A class inherits from another class using the ‘extends’ keyword. Child classes

inherit all properties and methods except private members and constructors

from the parent class.

class child\_class\_name extends parent\_class\_name

class Shape {

Area:number

constructor(a:number) {

this.Area = a

}

}

class Circle extends Shape {

disp():void {

console.log("Area of the circle: "+this.Area)

}

}

var obj = new Circle(223);

obj.disp()

Inheritance can be classified as −

Single − Every class can at the most extend from one parent class

Multi-level − The following example shows how multi-level inheritance works.

Multiple − A class can inherit from multiple classes.

TypeScript doesn’t support multiple inheritance.

Multiple Inheritance

Hierarchical Inheritance

Hybrid Inheritance

class Car {

Color:string

constructor(color:string) {

this.Color = color

}

}

class Audi extends Car {

Price: number

constructor(color: string, price: number) {

super(color);

this.Price = price;

}

display():void {

console.log("Color of Audi car: " + this.Color);

console.log("Price of Audi car: " + this.Price);

}

}

let obj = new Audi(" Black", 8500000 );

obj.display();

**typecast**

which allow you to convert a variable from one type to another type.

Type castings allow you to convert a variable from one type to another.

let str:any="test 234";

let s = <string>str;

let sq = str as number;

console.log(s);

console.log(sq);

TypeScript Type Assertion

is a mechanism which tells the compiler about the type of the variable.

Type assertion works like typecasting,

but it does not perform type checking or restructuring of data

The typecasting comes with runtime support,

whereas type assertion has no impact on runtime.

type assertions are purely a compile-time construct

TypeScript provides two ways to do Type Assertion. They are

Using Angular Bracket <>

Using as keyword

let ecode:any=344;

let empcode=<number>ecode;

let emmpc = ecode as string;

console.log(typeof(empcode));

Type Assertion with Object

Similarly, we might have a situation where we have an object that has been

declared without any properties yet

let employee = { };

employee.name = "John";

employee.code = 123;

interface Employee {

name: string;

code: number;

}

let employee = <Employee> { };

employee.name = "John";

employee.code = 123;

**static properties**

The static members can be defined by using the keyword static.

The static members of a class are accessed using the class name and dot

notation,

without creating an object

class Circle {

static pi: number = 3.14;

let pi1:number=90;

static calculateArea(radius:number) {

return this.pi \* radius \* radius;

}

}

console.log(Circle.pi)

console.log(Circle.calculateArea(5))

let ss= new Circle()

console.log(ss)

**abstract class**

Abstract class is a class which may have some unimplemented methods.

These methods are called abstract methods.

abstract class in Typescript using the abstract keyword.

Abstract classes are mainly for inheritance

We cannot create an instance of an abstract class.

An abstract class typically includes one or more abstract methods or property

declarations.

abstract class Person {

abstract name: string;

display(): void{

console.log(this.name);

}

}

class Employee extends Person {

name: string;

empCode: number;

constructor(name: string, code: number) {

super();

this.empCode = code;

this.name = name;

}

}

let emp: Person = new Employee("James", 100);

emp.display();