**What is TypeScript:**

TypeScript is an open-source programming language developed and maintained by Microsoft.

It is a strict syntactical superset of JavaScript, and adds optional static typing to the language.

TypeScript is designed for development of large applications and trans compiles to JavaScript.

**Why TypeScript is developed while having JavaScript?**

JavaScript development team introduced JavaScript as a client-side programming language.

But when people were using JavaScript then developer get to know that JavaScript can be used as a server-side programming language also.

But When JavaScript was growing then the code of JavaScript became complex and heavy.

Because of this, JavaScript was even not able to full fill the requirement of Object-oriented programming language.

This prevents JavaScript from succeeding at the enterprise level as a server-side technology.

Then TypeScript was developed by the development team to bridge this gap.

**Features of TypeScript:**

TypeScript Code is converted into Plain JavaScript Code

TypeScript code is not understandable by the browsers.

That’s why if the code is written in TypeScript then it is compiled and converted the code i.e. translate the code into JavaScript.

The above process is known as Trans-piled. By the help of JavaScript code, browsers are able to read the code and display.

**JavaScript is TypeScript:**

Whatever code is written in TypeScript can be converted to JavaScript by changing the extension from .ts to .js.

Use TypeScript anywhere: TypeScript code can be run on any browser, devices or in any operating system.

TypeScript is not specific to any Virtual-machine etc.

TypeScript supports JS libraries: With TypeScript, developers can use existing JavaScript code, incorporate popular JavaScript-

libraries, and can be called from other JavaScript code.

**Advantages**:

TypeScript always point out the compilation errors at the time of development only.

Because of this at the run-time the chance of getting errors are very less whereas JavaScript is an interpreted language.

TypeScript has a feature which is strongly-typed or supports static typing.

That means Static typing allows for checking type correctness at compile time. This is not available in JavaScript.

TypeScript is nothing but JavaScript and some additional features i.e. ES6 features.

It may not be supported in your target browser but TypeScript compiler can compile the .ts files into ES3, ES4 and ES5 also.

**Disadvantages:**

Generally, TypeScript takes time to compile the code.

TypeScript does not support abstract classes.

**Installation and Execution of TypeScript Files:**

TYPESCRIPT INSTALLATION & EXECUTION OF THE FILES

Step-1: Download & Install Node.js from:

<https://nodejs.org/downloads>

Step-2: Open Command Prompt and Install Typescript package by "npm" command:

**Run (in cmd):**

npm install -g typescript

Step-3: Check version of typescript by "tsc" Command:

**Run (to know the version):**

tsc -v

Step-4: Now Create new folder (Ex: TS is the folder name) into DESKTOP:

Step-5: Create new typescript with extension is ".ts" (example.ts) and save into TS-folder:

**example.ts**

console.log ("HELLO TYPESCRIPT");

Step-6: Now Activate TS-FOLDER, in Command Prompt by CD Command:

**Run (cd to typescript folder):**

cd TS

Step-7: Now Compile example.ts File by "tsc" command

**Run (to Compile):**

tsc example.ts

Step-7: Now Run example.js File by "node" command

**Run (To Run):**

node example.js

**Var, Let and Const:**

VAR, LET, CONST VARIABLES:

var and let and const are two relatively new types of variable declarations in JavaScript.

**var:**

var keyword "there is the possibility of overriding an existing value".

**let:**

let keyword "there is the no possibility of overriding an existing value".

**const:**

const statement values can be assigned once and they cannot be reassigned.

**Diff between var & let keyword Variables :**

1. var keyword variable is "function scope means both Global & Local Scope", But let keyword variable is "Block scope".

2. var keyword variable can be accessible anywhere in function scope & blocks Scope,

But let keyword variable Only be accessible inside the block scope where they are declared.

3. var keyword variable Re-declare the same variable in the same scope,

but let keyword variable Won’t redeclared. Then Show’s Uncaught Syntax Error: Identifier ‘variable' has already been declared.

**NEW DATATYPES:**

Whenever a variable is created, the intention is to assign some value to that variable but what type of value can be assigned to that

variable is dependent upon the datatype of that Variable.

In typescript, type System represents different types of datatypes which are supported by TypeScript.

**List of Datatypes:**

number : Both Integer as well as Floating-Point numbers

Boolean : Represents true and false

string : Represent a sequence of characters

null : used when an object does not have any value

undefined : Denotes value given to uninitialized variable

any : If variable is declared with any data-type then any type of value can be assigned to that variable

tuple : Tuple types is special array to allow you to express an array where the type of a fixed number of elements

is known, but need not be the same.

void : Generally used on function return-types, to declare a function that do not return a value, we can use void type.

Union : The ability to combine one or two types. Two or more data types are combined using the pipe symbol (|) to denote a Union Type. In other words, a union type is written as a sequence of types separated by vertical bars.

enum : user can also define its own data type. User-defined types include Enumerations (enums), classes, interfaces, arrays, and tuple.

**Template String Variable:**

Template strings are a small change to JavaScript in ES6(ECMAScript) but the convenience of multi-line strings and variable substitution is substantial. Template literals are string literals allowing embedded expressions.

**Multi Line String:**

In ES6 we have another way of defining strings, using the back-tick character  ( ` )

With ` strings can now span multiple lines and they are also formatted with new line characters.

let myStr = `my new string`;

let introduction = `My name is "Prashanth", I'm using single quote (') and double quote (") here`;

**Variable Substitution:**

Another really interesting feature of declaring strings with ` is that they can now expand variables using the ${ variable\_name }.

`My name is ${name}, and I'm ${age} years old`

**FOR OF LOOP :**

for..of and for…in loops give us a very clean and concise syntax to iterate over all kinds of iterables and enumerables like strings,

arrays and object literals.

for..of is a method, introduced in ES2015, for iterating over "iterable collections", like an array, strings, objects also

**Syntax**

for (variable of iterable)

{

// do stuff

}

**for of for array**

const array = ['a', 'b', 'c', 'd'];

for (const item of array) {

console.log(item)

}

// Result: a, b, c, d

|  |  |  |
| --- | --- | --- |
|  | **for...in** | **for...of** |
| **Applies to** | **Enumerable Properties** | **Iterable Collections** |
| Use with Objects? | Yes | No |
| Use with Arrays? | Yes, but not advised | Yes |
| Use with Strings? | Yes, but not advised | Yes |

**Look this Reference:**

https://alligator.io/js/for-of-for-in-loops/

**ARROW FUNCTION:**

Lambda refers to anonymous functions in programming. These are a concise mechanism to represent anonymous functions.

These functions are also called as Arrow functions.

Lambda Function – Anatomy: There are 3 parts to a Lambda function -

**Parameters**- A function may optionally have parameters

**The fat arrow notation/lambda notation (=>)** - It is also called as the goes to operator

**Statements** - represent the function’s instruction set.

**Lambda Expression:**

It is an anonymous function expression that points to a single line of code.

Syntax:

( [param1, parma2,…param n] )=>statement;

Lambda statement is an anonymous function declaration that points to a block of code. This syntax is used when the function body spans multiple lines.

Syntax:

( [param1, parma2…param n] )=>

{

//code Block

}

**Ex:(1)**

let sum = (x: number, y: number): number => {

return x + y; }

sum(10, 20); //returns 30

**Ex:(2)**

let Print = () => console.log("Hello TypeScript");

Print(); //Output: Hello TypeScript

**Ex:(3)**

let sum = (x: number, y: number) => x + y;

sum(3,4); //returns 7

**Ex:(4)**

class Employee {

empCode: number;

empName: string;

constructor(code: number, name: string) {

this.empName = name;

this.empCode = code;

}

display = () => console.log(this.empCode +' ' + this.empName)

}

let emp = new Employee(1, ‘Prashanth’);

emp.display();

**OPTIONAL PARAMETER FUNCTIONS**

Optional parameters can be used when arguments need not be compulsorily passed for a function’s execution.

A parameter can be marked optional by appending a question mark to its name.

The optional parameter should be set as the last argument in a function.

It’s works in Both Named and Named Less Functions also.

// Optional Parameters

sayHello(hello?: string)

{

console.log(hello);

}

sayHello(); // Prints 'undefined'

sayHello('world'); // Prints 'world'

**Rest Parameters**

Rest parameters don’t restrict the number of values that you can pass to a function.

However, the values passed must all be of the same type.

In other words, rest parameters act as placeholders for multiple arguments of the same type.

To declare a rest parameter, the parameter name is prefixed with three periods.

Any nonrest parameter should come before the rest parameter.

**Syntax:**

EX:(1)

function Greet(greeting: string, ...names: string[])

{

return greeting + " " + names.join(", ") + "!";

}

Greet("Hello", "Steve", "Bill"); // returns "Hello Steve, Bill!"

Greet("Hello");// returns "Hello !"

**Default Parameter**

Function parameters can also be assigned values by default. However, such parameters can also be explicitly passed values:

Ex

function test(x: number, y: number = 3): void

{

console.log(`x= ${x}, y=${y}`);

}

test(2);

test(2, 5);

**look This URL**

https://www.logicbig.com/tutorials/misc/typescript/function-optional-and-default-params.html

**Objects and Functions:**

class Person

{

     pname : String = `Prashanth`;

    walk()

    {

       console.log(`i'am Walking and i am `+this.pname);

    }

}

let a=new Person;

console.log(a.pname);

a.walk();

class Persons

{

    name: string;

    static id : number =100;  // static

    constructor(name: string)

    {

        this.name=name;

    }

    myfun()

    {

        console.log("Hi, I am " + this.name + "!");

    }

}

//OBJECT OF CLASS

let obj= new Persons("Prashanth");

console.log(Persons.id);

obj.myfun();

**Inheritance:**

**Single Inheritance:**

class Electronic

{

    constructor(mobilename :String ) {

        console.log('A new Mobile is Created '+mobilename);

    }

}

class Mobile extends Electronic

{

     constructor()

     {

        super("miA3");  // using super key word

    }

}

//OBJECT for CHILD CLASS

let obj = new Mobile();

**Example 2:**

class Person

{

    private name: string ;

    constructor(Name: string)

    {

        this.name = Name;

    }

    introduceSelf()

    {

        console.log("Hi, I am " + this.name + "!");

    }

}

class Friend extends Person

{

    yearsKnown: number;

    constructor(name: string, yearsKnown: number)

    {

        super(name);

        this.yearsKnown = yearsKnown;

    }

 timeKnown()

 {

        console.log("We have been friends for " + this.yearsKnown + " years.")

 }

}

 let p=new Friend("Prashanth" , 10);  //passing values

 p.introduceSelf();   // Output : Hi, I am Prashanth!

 p.timeKnown(); // Output : We have been friends for 10 years.

**Multi level Inheritance :**

class A

{

    a: number;

    constructor(a : number)

    {

        this.a=a;

    }

    methoda()

    {

        console.log("This method is from class A and the value of a is "+this.a );

    }

}

class B extends A

{

    b: any;

    constructor(a: number , b: any)

    {

        super(a);

        this.b = b;

    }

methodb()

{

    console.log("This method is from class B and the value of b is " +this.b);

}

}

class C extends B

{

    name: String;

    constructor(name : String ,a: number, b:any)

    {

        super(a,b);

        this.name=name;

        console.log(`name is `+this.name);

    }

    methodc()

    {

        console.log("values send from class C ")

    }

}

let x=new C("Prashanth" ,10,"Hello World!!");

x.methoda();

x.methodb();

x.methodc();

**Interfaces:**

TypeScript also have interfaces as we discussed in the beginning.

interface IVehicle

{

    start(type:string): string;

}

class Vehicle implements IVehicle

{

    constructor(public color:string){

    }

    start(type:string){

        return 'the ' + this.color + ' ' + type + ' started';

    }

}

class Car extends Vehicle

{

    constructor(color:string) {

        super(color);

    }

    start(){

        return super.start('car');

    }

}

interface ITrunk{

    openTrunk():void

}

interface IWindow{

    openWindow():void

}

class Sedan extends Car implements ITrunk,  IWindow{

    constructor(color:string) {

        super(color);

    }

    start(){

        return super.start() + ' and it is a Sedan';

    }

    openTrunk(){

        console.log('Trunk is open');

    }

    openWindow(){

        console.log('Window is open');

    }

}

class Truck extends Vehicle {

    constructor(color:string) {

        super(color);

    }

    start(){

        return super.start('truck');

    }

}

var car = new Car('green');

var sedan = new Sedan('red');

sedan.openTrunk();

sedan.openWindow();

console.log(sedan.start());

var truck = new Truck('blue');

console.log(truck.start());

**Output :**

PS D:\Repository\git\java-programing\TypeScript> tsc Interfaces.ts

PS D:\Repository\git\java-programing\TypeScript> node Interfaces.js

Trunk is open

Window is open

the red car started and it is a Sedan

the blue truck started

we can also create our own object types and use them

var person = {

    firstName: "Tom",

      lastName: "Hanks",

      sayHello:function() {

       }

};

 person.sayHello = function()

 {

    console.log("hello "+person.firstName);

    console.log("his full name is "+person.firstName+" "+person.lastName);

 }

 person.sayHello();

**Output :**

PS D:\Repository\git\java-programing\TypeScript> tsc objects.ts

PS D:\Repository\git\java-programing\TypeScript> node objects.js

hello Tom

his full name is Tom Hanks

**Exports and Imports :**

**export.ts file**

export var x = 123;

export function myfun ()

    {

            var x=10;

            var y=20;

            console.log(x+" "+y);

    }

    class Teacher {

        public work(): void {

            console.log('I am teaching');

        }

    }

    export default Teacher;

**import.ts file**

import {x} from './export';

console.log(x);

import {myfun} from './export';

myfun();

import Teacher from './export';

var obj=new Teacher();

obj.work();

we can export variables, methods, classes and even interfaces also

**export.ts :**

interface IWork

    {

        work(): void;

    }

    export default IWork;

**import.ts :**

import IWork from './export';

class Lecturer implements IWork {

    public tenure: boolean;

    constructor() {

        this.tenure = true;

    }

    public work(): void {

        console.log("I am teaching");

    }

}

var obj=new Lecturer();

obj.work();

we can export and import multiple functions too

export.ts file :

    function square(x) {

        return Math.pow(x,2)

    }

    function cow() {

        console.log("Mooooo!!!")

    }

    export {square, cow};

import.ts

import {square, cow} from './export';

console.log(square(2));

cow();