**Angular**

Typescript:-

* It is Programming language
* Why Typescript :-Agnular2+ (Angular 2 To Angular 10)
* Who Released Typescript:- Microsoft
* It is object oriented programming language
* Typescript is superset of Java script(Advanced JavaScript)
* Browser don’t understand Typescript , we need to convert into Javascript
* Transpilation is a technique to convert Typescript into equivalent Javascript using the tool called TSC(typescript compiler tool).
* It is save using .Js extension.
* How to install Typescript:- using npm install –gtypescript@latest
* Npm stands for Node Packaging manager and npm is present in Node, so we need to Install Node

Question :- If javascript is there then why we need typescript.

Javascript doesn’t follow OOPS concept , to follow we need to write thousands of code , that’s why we go for typescript because it follow OOPS concept and compiler automatically generates thousands of line of code while converting to javascript.

/\*

    variables

        - variables are used to store the data.

        - by using variables we can store any type of data.

        - we will declare variables by using "var" / "let" and "const" keyword

    Syntax

    ------

        var/let/const variablename:datatype = value;

\*/

//number datatype

var var1:number = 100;          //decimal

var var2:number = 100.12345;    //float or double

var var3:number = 0x123ABC;     //hexadecimal number

var var4:number = 0o123;        //octal number

var var5:number = 0b1010;       //binary number

console.log(var1,var2,var3,var4,var5);

//string datatype

//""  ,  ''  and ``(backtick)

//``(backtick) operator introduced in ES6

//``(backtick) operator used to define the paragraphs

var sub\_one:string = "Angular10";

var sub\_two:string = 'NodeJS';

var sub\_three:string = `MongoDB`;

var mean:string = `${sub\_one}<==>${sub\_two}<==>${sub\_three}`;

console.log(mean);

var tbl\_name:string = "employees";

var sal:number = 100000;

var query1:string = `select \* from ${tbl\_name} where esal>${sal}`;

console.log(query1);

var login\_tbl:string = "login\_details";

var u\_name:string="admin";

var u\_pwd:string="admin";

var query2:string = `select \* from ${login\_tbl} where uname='${u\_name}' and upwd='${u\_pwd}'`;

console.log(query2);

//union type

var data:number|string = "100";

data = 100;

console.log(data);

//boolean

//true or false

var flag:boolean = true;

console.log(flag);

//Any data type

//used to store any value

var my\_value:any=`Angular`

//array of number

var num\_value:Array<number>=[10,20,30,40,50]

var num\_value1:number[]=[60,70,80,90,100]

num\_value.forEach((element,index)=>{

    console.log(element,num\_value1[index])

})

//array of string

var str:Array<string>=['Angular','Nodejs','MongoDB']

var str1:string[]=  ['Java','.net','SQL Server']

str.forEach((element,index)=>{

    console.log(element,str1[index])

})

//Json

//Stand for Javascript object notation

//It is also called javascript Object

//Json is used to transfer the data over the network

//Data type of jsaon is 'any'

//key & value are seprated by ':'

//key and value pairs are seprated by ','

var obj:any={

    "first\_value":"Angular",

    "second\_value":"NodeJs",

    "third\_value":"MongoDB"

}

console.log(obj.first\_value,obj.second\_value,obj.third\_value)

//Json Array

var products:Array<any>=[

    {"p\_Id":111,"P\_Name":"P\_one","p\_cost":1000},

    {"p\_Id":222,"P\_Name":"P\_two","p\_cost":2000},

    {"p\_Id":333,"P\_Name":"P\_three","p\_cost":3000},

    {"p\_Id":444,"P\_Name":"P\_four","p\_cost":4000}

]

products.forEach((element,index)=>{

    console.log(element.p\_Id,element.P\_Name,element.p\_cost)

})

//Tuple

//Collection of heterogenious element called as tuple

//data type of tuple is any

var T\_arr:any=['Angular',10,true,{"sub":'NodeJs'}]

console.log(T\_arr);

T\_arr.push('MongoDB');

console.log(T\_arr);

T\_arr.unshift('NareshIT');

console.log(T\_arr);

T\_arr.pop();

console.log(T\_arr);

T\_arr.shift();

console.log(T\_arr);

var[v1,v2,v3,v4]=T\_arr;

console.log(v1,v2,v3,v4.sub);

//let keyword

//introduced in  java script ES5

// let is a variable used to store any data

// let solve the problem occured due to var keyword

for(var i:number=0;i<5;i++)

{

};

console.log(i);  //We are expecting error

// var : 5 , let : Cannot find name 'i'.

var x:number=100;

{

    console.log(x);

    var x:number=200;

}

//issue is called global polluting issue, because local variable effect the global variable

console.log(y);

var y:number=100; //var : undefined(variable hoisting) let:Block-scoped variable 'y' used before its declaration.

/\*

for(let i:number=0;i<5;i++){

setTimeout(()=>{

console.log(i);

},5000);

}; //var : 5 5 5 5 5 //let : 0 1 2 3 4

\*/

/\*

function fun\_one(){

var var\_one:number = 100;

{

let var\_two:number = 200;

}

console.log(var\_two); //Cannot find name 'var\_two'.

};

fun\_one();

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*

var let

introduced in ES1 introduced in ES5

variable hoisting we can overcome by using

raised because of let keyword

var keyword

global polluting issue we can overcome global polluting

raised because of var issue by using "let" keyword

keyword

var members are not the let members are block scoped

block scoped members members

behavour of "var" and "let" behavour of "var" and "let"

is same in function call is same in function call

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

//const

//const keyword introduced in ES5

//const keyword also used to declare the variables

//reinitilization not possible by using const keyword

/\*

const data1:number = 100;

data1 = 200; //Cannot assign to 'data1' because it is a constant.

\*/

var obj = {

val: 10

};

console.log(obj); //{ val: 10 }

//obj = {}; //Cannot assign to 'obj' because it is a constant.

obj.val = 20;

console.log(obj);

console.log(100 == "100"); //true

console.log( 100 === "100" ); //false

console.log( 1 == "one" ); //false

//number //string //boolean //undefined //null //bigint //Symbol

var x;

console.log(x); //undefined

x = null;

console.log(x); //null

//2^53 to -2^53

var y=1234512345123451234512345123451234512345123451234512345123451234512345123451234512345n;

console.log(y);

//typeof() function is used to find the datatype

* console.log( typeof 100 ); //number
* console.log( typeof "100" ); //string
* console.log( typeof 100.12345 ); //number
* console.log( typeof 0x123ABC ); //number
* console.log( typeof 10n ); //bigint
* console.log( typeof undefined ); //undefined
* console.log( typeof null ); //object
* console.log( typeof [] ); //object
* console.log( typeof true); //boolean
* console.log( typeof console ); //object
* console.log( typeof {} ); //object
* console.log( typeof Date ); //function
* console.log( typeof let ); //undefined
* console.log( undefined == undefined );
* console.log( undefined === undefined );
* console.log( null == null );
* console.log( null === null );
* console.log( undefined == null );
* console.log( undefined === null ); //false

console.log(0.1+0.2 == 0.30000000000000004); //false 64-bit memory algorithm (more accuracy)

/\*

collection of indexed elements called as array

\*/

/\*

1) map()

- manipulate the each and every array element

console.log(

[10,20,30,40,50].map((element,index)=>{

return element\*10;

})

); //[ 100, 200, 300, 400, 500 ]

console.log(

[1,2,3,4,5].map((element,index)=>{

return element+"$"

})

); //[ '1$', '2$', '3$', '4$', '5$' ]

\*/

/\*

2) filter()

console.log(

[10000,20000,30000,40000,50000].filter((element,index)=>{

return element>=30000;

})

); //[ 30000, 40000, 50000 ]

console.log(

[1,2,3,4,5].map((element,index)=>{

return element\*100;

}).filter((element,index)=>{

return element>=200;

})

); //[ 200, 300, 400, 500 ]

\*/

/\*

3) reduce()

- it is used to find the sum of array elements

console.log(

[1,2,3,4,5].reduce((firstElement,nextElement)=>{

return firstElement+nextElement;

})

); //15

console.log(

[1,2,3,4,5].map((element,index)=>{

return element\*1000;

}).filter((element,index)=>{

return element>=3000;

}).reduce((firstValue,nextValue)=>{

return firstValue+nextValue;

})

); //12000

\*/

/\*

4) reduceRight()

console.log(

["Angular10","to","welcome"].reduceRight((firstValue,nextValue)=>{

return firstValue+" "+nextValue;

})

); //welcome to Angular10

\*/

/\*

let arr1 = [10,20,30,40,50,60,70,80,90,100];

arr1.splice(4,2);

console.log(arr1);

\*/

/\*

let arr1 = [1,2,3,4,5];

console.log( arr1.fill(100)); //[ 100, 100, 100, 100, 100 ]

console.log( arr1.fill(200,2)); //[ 100, 100, 200, 200, 200 ]

console.log( arr1.fill(300,0,3)); //[ 300, 300, 300, 200, 200 ]

\*/

/\*

console.log(

[100,200,300,400,500].every((element,index)=>{

return element>100;

})

);

\*/

/\*

console.log(

[10000,20000,30000,40000,50000].some((element,index)=>{

return element>50000;

})

);

\*/

let arr1 = [10,20,30,40,50,60,70,80,90,100]

console.log(arr1.copyWithin(1));

let arr2 = [10,20,30,40,50,60,70,80,90,100]

console.log( arr2.copyWithin(5) );

let arr3 = [10,20,30,40,50,60,70,80,90,100]

console.log( arr3.copyWithin(4,5) );

let arr4 = [10,20,30,40,50,60,70,80,90,100]

console.log( arr4.copyWithin(2,4,7) );

/\*

let arr = [10,20,30,40,50,60,70,80,90,100];

arr.splice(4,2);

console.log(arr); //[10,20,30,40,70,80,90,100]

arr.splice(4,1);

console.log(arr); //[10,20,30,40,80,90,100]

arr.splice(-6,2);

console.log(arr); //[ 10, 40, 80, 90, 100 ]

arr.splice(-1,1);

console.log(arr); //[ 10, 40, 80, 90 ]

arr.splice(1,0,20,30);

console.log(arr); //[ 10, 20, 30, 40, 80, 90 ]

arr.splice(-2,0,50,60,70);

console.log(arr); //[10, 20, 30, 40, 50,60, 70, 80, 90]

arr.splice(-1,1,90,100);

console.log(arr); //[10, 20, 30, 40, 50,60, 70, 80, 90,100]

\*/

/\*

let arr = [10,100,1000,10000,20,200,2000,20000];

console.log(arr.findIndex((element,index)=>{

return element === 20;

})); //4

console.log(arr.findIndex((element,index)=>{

return element === 30000;

})); //7 //-1

arr.splice( arr.findIndex((element,index)=>{

return element === 1000;

}),1);

console.log(arr);

arr.splice(arr.findIndex((element,index)=>{

return element === 20;

}),1);

console.log(arr);

let arr1 = [{"val":10000},

{"val":20000},

{"val":30000},

{"val":40000},

{"val":50000}];

arr1.splice(arr1.findIndex((element,index)=>{

return element.val === 30000;

}),1);

console.log(arr1);

\*/

/\*

indexOf()

\*/

/\*

let arr = [10,20,30,10,30,20,50,10,20,40];

arr.forEach((element,index)=>{

console.log( arr.indexOf(index) );

});

let arr1 = [73,56,29,48,64,93,56,29,73];

arr1.forEach((element,index)=>{

console.log( arr1.indexOf(element) );

});

let arr = [10,20,10,30,20,30];

console.log(

arr.filter((element,index)=>{

return arr.indexOf(element) == index;

})

); //[ 10, 20, 30 ]

let arr = [10,100,1000,10000,20,200,2000,20000,30,300];

console.log(

arr.slice(4,6)

); //[ 20, 200 ]

console.log(

arr.slice( arr.findIndex((element,index)=>{

return element === 20;

}),arr.findIndex((element,index)=>{

return element === 2000;

}))

); //[ 20, 200 ]

\*/

console.log(

[10,20,30].reverse()

); //[ 30, 20, 10 ]

console.log(

["Angular","NodeJS","ReactJS"].reverse()

); //[ 'ReactJS', 'NodeJS', 'Angular' ]

//['SJtcaeR','SJedoN','ralugnA']

//['ralugnA','SJedoN','SJtcaeR']

console.log(

Array.from("Hello").reverse().join("\*NareshIT\*")

); //olleH

//[1,2,3,4,5,6,7,8,9,10]

//[6,7,8,9,10,1,2,3,4,5]

//[40,15]

Class Concept:-

/\*

class

- collection of variables and functions called as class.

- we will create classes by using "class" keyword.

- we will create the object the classes by using "new" keyword.

- we will define the constructor by using "constructor" keyword.

- recomended modifier for variables is "private" because of secuirity.

- recomended modifier for functions is "public"

\*/

/\*

class class\_one{

private sub\_one:string;

private sub\_two:string;

private sub\_three:string;

constructor(){

this.sub\_one = "Angular10";

this.sub\_two = "NodeJS";

this.sub\_three = "MongoDB";

};

public getSubOne():string{

return this.sub\_one;

};

public getSubTwo():string{

return this.sub\_two;

};

public getSubThree():string{

return this.sub\_three;

};

};

let obj:class\_one = new class\_one(); //Duck Typing

console.log( obj.getSubOne(),

obj.getSubTwo(),

obj.getSubThree() );

\*/

class class\_one{

constructor(public arg1:string,

public arg2:string,

public arg3:string){}

};

let obj1:class\_one = new class\_one("Angular10","NodeJS","MongoDB");

console.log(

obj1.arg1,

obj1.arg2,

obj1.arg3

);

let obj2:class\_one = new class\_one("ReactJS","NodeJS","MySQL");

console.log(

obj2.arg1,

obj2.arg2,

obj2.arg3

);

let obj3:class\_one = new class\_one("VueJS","NodeJS","MySQL");

console.log(

obj3.arg1,

obj3.arg2,

obj3.arg3

);

Inheritance

/\*

class class\_one{

public fun\_one():string{

return "fun one !!!";

};

};

class class\_two extends class\_one{

public fun\_two():string{

return "fun two !!!";

};

};

let obj:class\_two = new class\_two();

console.log( obj.fun\_one(),

obj.fun\_two() ); //fun one !!! fun two !!!

\*/

/\*

class class\_one{

public mean():string{

return "MEAN Stack Development";

}

};

class class\_two extends class\_one{

public mern():string{

return "MERN Stack Development Soon...!";

}

};

class class\_three extends class\_two{

public mevn():string{

return "MEVN Stack Development Soon...!"

};

};

let obj:class\_three = new class\_three();

console.log( obj.mean(),

obj.mern(),

obj.mevn() );

\*/

/\*

class class\_one{}

class class\_two{}

class class\_three extends class\_one,class\_two{}

\*/

/\*

class class\_one{}

class class\_two extends class\_one{}

class class\_three extends class\_one{}

class class\_four extends class\_two,class\_three{}

\*/

/\*

class class\_one{}

class class\_two extends class\_one{}

class class\_three extends class\_one{}

class class\_four extends class\_one{}

\*/

/\*

1) single level inheritance

2) multi level inheritance

3) multiple inheritance

4) hybrid inheritance

5) hierarchical inheritance

\*/

Polymorphism

/\*

// Overloading concept

class class\_one{

public my\_fun(arg1:number,arg2:number):void;

public my\_fun(arg1:string,arg2:string):void;

public my\_fun(arg1:boolean,arg2:boolean):void;

public my\_fun(arg1:any,arg2:any):void{

console.log(arg1,arg2);

}

};

new class\_one().my\_fun(10,10);

new class\_one().my\_fun("Hello\_1","Hello\_2");

new class\_one().my\_fun(true,true);

new class\_one().my\_fun(10,"Hello");

\*/

/\*

Overriding concept:-

class class\_one{

public dbFun():string{

return "mysql data soon...!";

};

};

class class\_two extends class\_one{

public dbFun():string{

return "mongodb data soon...!";

};

};

console.log( new class\_two().dbFun() );

\*/

// when there is not any constructor in child class parent class constructor will be called

And if child class constructor is there then we have to use super keyword

class class\_one{

constructor(public wish:string){}

};

class class\_two extends class\_one{

constructor(public arg1:string){

super(arg1);

}

};

let obj:class\_two = new class\_two("Hello....!");

console.log( obj.wish );