TypeScript

let sales= 123\_456\_789

let course:string= "Typescript"

let is\_published=true

let level;

level=1;

//Array in Typescript

let marks=[20,20,30,40,50]

for(var i=0; i<marks.length; i++)

{

    console.log("Marks obtained by all student " +(i+1)+ " = " +marks[i])

}

//let fruits1: string[] = ['Apple', 'Orange', 'Banana'];

let fruits2: Array<string>= ['Apple', 'Orange', 'Banana']

for(var index in fruits2)

{

    console.log(fruits2[index]);  // output: Apple Orange Banana

}

/\*for(var i = 0; i < fruits.length; i++)

{

    console.log(fruits[i]); // output: Apple Orange Banana

}\*/

let fruits3: Array<string>;

fruits3 = ['Apple', 'Orange', 'Banana'];

let ids: Array<number>;

ids = [23, 34, 100, 124, 44];

let values1: (string | number)[] = ['Apple', 2, 'Orange', 3, 4, 'Banana'];

// or

let values2: Array<string | number> = ['Apple', 2, 'Orange', 3, 4, 'Banana'];

//Tuple

//TypeScript introduced a new data type called Tuple.

//Tuple can contain two values of different data types.

var empId: number = 1;

var empName: string = "Steve";

// Tuple type variable

var employee1: [number, string] = [1, "Steve"];

console.log("Employee 1 details = " + employee1[0] + ", " +employee1[1] )

var employee2: [number, string] = [2, "Bill"];

console.log("Employee 2 details = " + employee2[0] + ", " +employee2[1] )

//A tuple type variable can include multiple datatypes as follows-

var employee: [number, string] = [1, "Steve"];

var person: [number, string, boolean] = [1, "Steve", true];

var user: [number, string, boolean, number, string];// declare tuple variable

user = [1, "Steve", true, 20, "Admin"];// initialize tuple variable

//You can declare an array of tuple also

var emp: [number, string][];

emp = [[1, "Steve"], [2, "Bill"], [3, "Jeff"]];

//Enums

//enums allow us to declare a set of named constants i.e. a collection of related values that can be numeric or string values.

enum Resource{Printer='p', Scanner='s', USBDrive='u', TapeDrive='t', CPU='c' }

console.log(Resource.Scanner);

//Functions

//1.Named Function

function display() {

    console.log("Hello TypeScript!");

}

display();

function Sum1(x: number, y: number) : number {

    return x + y;

}

let result=Sum1(2,3);

 console.log(result);

 //2. Anonymous Function

let greeting = function() {

    console.log("Hello TypeScript!");

};

greeting();

let Sum2 = function(x: number, y: number) : number

{

    return x + y;

}

console.log(Sum2(2,3));

//TypeScript - Arrow Functions

//Fat arrow notations are used for anonymous functions i.e for function expressions.

//They are also called lambda functions in other languages.

//Syntax: (param1, param2, ..., paramN) => expression

let sum3 = (x: number, y: number): number =>{

    return x+y;

}

console.log(sum3(10, 20));

let greeting1= ()=> console.log("Arrow Function in TypeScript")

 greeting1();

//Typescript class

/\*A class can include the following:

   1. Constructor

   2. Properties

   3. Methods\*/

    class Employee {

        empCode: number;

        empName: string;

        constructor(code: number, name: string) {

                this.empName = name;

                this.empCode = code;

        }

        getSalary() : number {

            return 10000;

        }

        display():void {

            console.log("Name = " + this.empName +  ", Employee Code = " + this.empCode + ", Salary = " + this.getSalary());

        }

    }

    let emp1 = new Employee(100,"Steve");

   emp1.display();