TypeScript

Code format: shift + Alt + F

var vs let vs const

var

- we can use var keyword anywhere in ts file.
- We can use var variable before declaration such as
  console.log(x);
  var x = 10;
- We can declare same variable name with same datatype any number of time using var keyword such as
  var x:string;
  var x:string;

let

- We can use let keyword anywhere in ts file.
- We can not use let variable before it's declaration such as
  console.log(x);
  var x = 10;
  Note: This is wrong.
- We can not declare same variable name with same datatype any number of time using let keyword such as
  var x:string;
  var x:string;
  Note: this is wrong
- But We can declare let variables in different block with same name with same datatype because it's scope is different such as

```
If(a>0){
   let x:string;
}else{
  let x:string;
}
```

Note: This is the main difference between var and let keyword.

const

- const keyword is like let keyword as scope level.
- We can not declare same variable name with same datatype any number of time using let keyword such as let keyword.
- We can declare and assign the value of declared variable in single line. But after assigned the value for const variable we can't change it.
- We can declare and assign the value of const variable outside the block also.

TypeScript Types

1. Build in or Primitives Datatypes
   - Number
   - String
   - Void
   - Null
   - Boolean
2. User -Defined Datatypes
   - Array
   - Touple
   - Class
   - Enum
   - Functions
   - Interface

null

how we can define a variable as a null type.
var a: null;
how we can assign a variable as a null.
var str: string=null;

undefined

var a:undefined;
a=10; //will give error becoz we have made a as a undefined type so we can't assign any values other than undefined.

var b: number=undefined;
b=10; // will not give any error we can assign any values of any type here. Becoz we have made b as a number type not undefined.

Array in TypeScript

```
// traditional way to define array
//way - 1

var list: number[]=[2,4,5];
console.log(list);
console.log(list[2]);

var str: string[]=['adarsh','chauhan'];
console.log(str);
console.log(str[1]);

var allTypeValues:any[]=['adarsh','chauhan',3,4,6,true,3.6];
console.log(allTypeValues);
console.log(allTypeValues[6]);

//way - 2

var newList:Array<number>=[3,5,7,9];
console.log(newList);
console.log(newList[3]);
```

Tuple

If we want to allow only particular types of elements are stored in Array not all types of elements then we must use Tuple in typescript such as :

Difference between tuple and any type is
any: any type will allow to store all types of datatypes values in Array such as

```
var x: any=[1,2,'abc','chauhan',2.5,true];
```

touple: only selected values of datatype we can store in array which has given at the time of declaration such as

```
var x: [number,string,boolean]=[10,'abc',true];
var x:[number,string,boolean]=['abc',true,10];  //will give error becoz datatype order not matched
```

Note: but here passing values of datatypes must be in same order in which datatypes are declared.

Define array using tuple

```
//define array in tuple
var x:[number,string,boolean];
x=[10,'abc',true];
console.log(x);
console.log(x[2]);
//define array of array in tuple
var y:[number,string,boolean][];
y=[[10,'abc',true],[20,'xyz',false],[30,'LML',true]];
console.log(y);
console.log(y[2][1]);

//if we want to add new element in array then use push() method it will not replace existing values.
var employee:[number, string];
employee=[10,'adarsh chauhan'];
employee.push(12,'Ramesh Tyagi');

console.log(employee);
```

Interface

```
// define variables or key-pairs values in interface

interface keyValuepair{
    key:string;
    value:string;
}

//implement or use interface
function show(){
    let keyvaluespair1:keyValuepair={key:'eid',value:'emp1'};
    let keyvaluespair2:keyValuepair={key:1,value:'emp1'};  // will give error becoz key is string type
    let keyvaluespair3:keyValuepair={key:'eid',value:2};  // will give error becoz value is string type
    let keyvaluespair3:keyValuepair={value:'emp1', key:'eid'};  // will also give error becoz value will come at
second place as it is defined in interface

    console.log(keyvaluespair1);
}

// define methods in interface

interface Calculator{
    sum(num1:number,num2:number):number;
}

var add:Calculator={
    sum(num1:number,num2:number){
        return num1+num2;
    }
}

console.log("addition : "+add.sum(10,20));
```

enum

```
//here indexing will start from zero
enum color {
    RED,
    YELLOW,
    PINK
}
//     or now here indexing will start from 50
enum color2 {
    RED = 50,
    YELLOW,
    PINK
}
//     or now here indexing will start from given values
enum color3 {
    RED = 100,
    YELLOW=150,
    PINK=200
}

let myColor = color.RED;
console.log(myColor);

let myColor2 = color2.RED;
console.log(myColor2);

let myColor3 = color3.PINK;
console.log(myColor3);
```

function

functions are of two types:

1. Named function
2. Anonymous function

1. **Named function**

```
//define function with return type in function signature
function addition():number{
    let z: number = 30;
    document.write('z : '+z);

    return z;
}

//define function with no return type in function signature
function additionWithoutReturnType(){
    let z:number = 30;
    document.write('z : '+z);

    return z;
}


addition();
additionWithoutReturnType();
```

2. **Anonymous function**

This function doesn't have function name and after creating this function, it will store into some variable, now this variable will act as a function and we can call this function with variable name (means variable name is function name) like below:

```
// anonymous function

let addition = function():number{
    let x = 20;
    let y = 30;
    return x+y;
};

console.log('add : '+addition());

//parameterized
let num1:number = 0;
let num2:number = 0;;

let addition2 = function(num1:number,num2:number):number{
    this.num1 = num1;
    this.num2 = num2;
    return num1 + num2;
}

console.log(addition2(150,100)+` is the addition of ${num1} and ${num2}`);
```

Generic Type

```
function show(agr) {
    return agr;
}

// or using generic type

function showGeneric<T>(agr:T):T {
    return agr;
}

var op1 = showGeneric<string>('sahosoft');
var op2 = showGeneric<number>(100);

document.writeln('<br>');
document.writeln('op1 : ' + op1);
document.writeln('<br>');
document.writeln('op2 : ' + op2);
```

for

```
//types of for loop
/**
 *    1. traditional for loop
 *          ex:
 *                for(let i=0;i<10;i++){
 *
 *                }
 *    2. in for loop, this loop will give index of an array
 *          ex:
 *                var arr = [10,20,30,40,50];
 *                for(let i in arr){
 *                   console.log(i);
 *                   console.log(arr[i]);
 *                }
 *
 *    3. of for loop, this loop will give val of an array
 *          ex:
 *                var arr = [10,20,30,40,50];
 *                for(let val of arr){
 *                   console.log(val);
 *                }
 *  */
```

Passing default parameter in function

We can pass default value of parameter in function and at the time of calling this function we can skip the value of default parameter to be passed in function. If we have two parameters then we can make either one or all parameters as default parameter so at the time of calling this function we can skip the value of default parameters to be passed in function. See the below example for it.

```
// without default parameter
function add(num1:number,num2:number):number{
    return num1+num2;
}

console.log(add(10,20));  // output : 10 + 20 = 30


// with default parameter

function addition(num1:number,num2:number=40):number{
    return num1+num2;
}

console.log(addition(10));

// output : 50   //using 10 + 40 = 50

// passing default value in parameter and also passing value in argument.
function additional(num1:number,num2:number=40):number{
    return num1+num2;
}

console.log(additional(10,20));   // output : 10 + 20 = 30
```

optional parameter in function

1. optional parameter is used for passing value as optional for any variable but it will assign 'undefined' value for that variable.
2. We can add optional variable at the last in function as a parameter.
3. We can add one or more optional parameters in function.
4. We can denote optional parameter by using ' ? 'symbol.
5. We must place ? symbol just after variable name and before colon symbol of variable datatype.
6. We can't add optional parameters at first place in function as a parameter.
7. If we don't pass value of optional parameter then it's value will be 'undefined'.
8. If we pass value of optional parameter then it's value will be considered which we have given in function calling.

```
// case-1 ---> if we don't pass value for optional parameter variable.
function show(num1:number,num2?:number):number{
    if(num2==undefined){
        num2 = 10;
    }
    return num1+num2;
}

console.log('num1 + num2 : '+show(10));

// case-2 ---> if we pass value for optional parameter variable.
function show2(num1:number,num2?:number):number{
    if(num2==undefined){
        num2 = 10;
    }
    return num1+num2;
}

console.log('num1 + num2 : '+show(10,100));
```

Rest Parameter

1. If don't know how many arguments are coming while calling any function then there we can use Rest parameter concept.
2. We can denote rest parameter symbol with 3 dots (…) and this symbol will come Infront of variable name and make this variable as a array.
3. We can't add rest parameter at first parameter of any function if there are at least one more parameter exist already.
4. It we can pass only optional parameter also inside function as a parameter.

```
//case-1 ----> passing mutiple arguments from calling function

function show(str1:string,...str2:string[]):string{
    return str1+' and '+ str2;
}
console.log(show('A','B','C','D'));

//case-2 ----> passing mutiple arguments from calling function if there is only rest parameter

function show2(...str2:string[]):string{
    return str2.toString();   // becoz str2 is retruning array
}
console.log(show('A','B','C','D'));

// array nottation to define and call function
let show3 = (...str2:string[]):string=>{
    return str2.toString();   // becoz str2 is retruning array
};
console.log(show('A','B','C'));
```