Call signature Interface.

Interface \_\_\_\_\_\_

{

(x:number):number;

}

If we declare a variable of type \_\_\_\_\_ then we can assign as address of the function to it or type (x:number):number means a function which will take input of type number and give output of type number. (as per the given condition)

Because in underline javascript function is also an object.

interface AdderCallSingature

{

(e: number, f: number): number;

}

function justDoIt(x:number, y:number):number

{

return x+y;

}

function eg1(){

let adder: AdderCallSingature;

adder = just do it;

let result: number;

result = adder(10,20);

console.log(result);

}

eg1();

But if we pass more than required. (lets try to break the guideline)

interface AdderCallSingature

{

(e: number, f: number): number;

}

function justDoIt(x: number, y: number, z: number): number

{

return x+y+z;

}

function eg2(){

let adder: AdderCallSingature;

adder = justDoIt; // incorrect

let result : number;

result = adder(10,20, 30); // incorrect

console.log(result);

}

eg2();

/\*

Line 12: Type '(x: number, y: number, z: number) => number' is not assignable to type 'AdderCallSingature'.

Target signature provides too few arguments. Expected 3 or more, but got 2.

Line 14: Expected 2 arguments, but got 3.

\*/

As we know in JavaScript, we can add properties dynamically.

But there is no control in js

But with the help of us, we can control it.

**Function names will become pointers internally.**

In javascript we know a function is an object

So we can add properties dynamically

Function abcd(){

}

Abcd.xyz = 30;

Call signature interface.

Bulbfactory se declared variable ek pointer hoga jo ek esse function ka address store kar sakta he jis ka ye(anything) parameter and wo(anything) return type ho.

BUT BUT

But that function must have property of type number and name productionCount.

**productionCount:number**

interface Bulb{

wattage:number;

printWattage():void;

}

// following is call signature interface

interface BulbFactory{

productionCount:number;

(wattage:number):Bulb;

}

function philipsBulbfactory(w:number):Bulb

{

if(w<0 || w>240) return null;

let b:Bulb;

b = {

wattage:w,

printWattage():void{

console.log(`Wattage is ${w}`);

}

};

philipsBulbfactory.productionCount++;

return b;

}

philipsBulbfactory.productionCount = 0;

function eg3(){

let f:BulbFactory;

f = philipsBulbfactory;

let bulb1:Bulb;

bulb1=f(-20);

console.log(bulb1);

let bulb2:Bulb;

bulb2=f(100);

console.log(bulb2);

bulb2.printWattage();

let bulb3:Bulb;

bulb3=f(200);

console.log(bulb3);

bulb3.printWattage();

console.log(`Number of bulbs produced nu Philips bulb factory ${philipsBulbfactory.productionCount}`);

console.log(`Number of bulbs produced nu Philips bulb factory ${f.productionCount}`);

}

eg3();

If we will not give productioncount

interface Bulb{

wattage:number;

printWattage():void;

}

// following is call signature interface

interface BulbFactory{

productionCount:number;

(wattage:number):Bulb;

}

function philipsBulbfactory(w:number):Bulb

{

if(w<0 || w>240) return null;

let b:Bulb;

b = {

wattage:w,

printWattage():void{

console.log(`Wattage is ${w}`);

}

};

return b;

}

function eg4(){

let f:BulbFactory;

f = philipsBulbfactory;

}

eg4();

/\*\*

\* Property 'productionCount' is missing in type '(w: number) => Bulb' but required in type 'BulbFactory'.

\*/

If we try to create additional property

We can not add additional property through the interface pointer.

But in function we can do it.

interface Bulb{

wattage:number;

printWattage():void;

}

// following is call signature interface

interface BulbFactory{

productionCount:number;

(wattage:number):Bulb;

}

function philipsBulbfactory(w:number):Bulb

{

if(w<0 || w>240) return null;

let b:Bulb;

b = {

wattage:w,

printWattage():void{

console.log(`Wattage is ${w}`);

}

};

philipsBulbfactory.productionCount++;

return b;

}

philipsBulbfactory.productionCount = 0;

philipsBulbfactory.yearOfEstablishment = 2003;

function eg5(){

let f:BulbFactory;

f = philipsBulbfactory;

console.log(`Number of bulbs producted nu philips bulb factory ${f.productionCount}`);

// console.log(`yes of estb is ${f.yearOfEstablishment}`); // incorrect

console.log(`yes of estb is ${philipsBulbfactory.yearOfEstablishment}`); // incorrect

}

eg5();

/\*

Property 'yearOfEstablishment' does not exist on type 'BulbFactory'.

// we can not access additional property thouth interface pointers.

\*/

If we make a variable of type interface

We can only define the properties that we mentioned in the interface

Now we want someone can add n number of property but that type should be string and it can only accept the number type value.

interface StringNumberKeyValue

{

[x:string]:number;

}

function eg6()

{

let kv:StringNumberKeyValue;

kv={};

kv.bus=5000000;

kv.cycle=2000;

kv.car=1000000;

console.log(kv);

// or

kv={

bus:5000000,

cycle:2000,

car:1000000

}

}

eg6();

If we try to add anything else like

Value of type string and value also string

// or

kv={

bus:5000000,

cycle:2000,

Car:1000000

Cost: “$200”; //// incorrect // Type 'string' is not assignable to type 'number'.

}

Error

/\*

- error TS2322: Type 'string' is not assignable to type 'number'.

17 cost: "$200"

~~~~

eg8.ts:3:5

3 [x:string]:number;

~~~~~~~~~~~~~~~~~~

The expected type comes from this index signature.

\*/

Also read the message the expired type come from this **index signature**

Now; try to define more property of type string but with value string.

interface StringNumberKeyValue

{

[x:string]:number;

[y:string]:string;

}

Error

/\*

eg9.ts:3:5 - error TS2374: Duplicate index signature for type 'string'.

3 [x:string]:number;

~~~~~~~~~~~~~~~~~~

eg9.ts:4:5 - error TS2374: Duplicate index signature for type 'string'.

4 [y:string]:string;

~~~~~~~~~~~~~~~~~~

Now try with different type type number value string

interface StringNumberKeyValue

{

[x:string]:number;

[y:number]:string;

}

function eg10()

{

let kv:StringNumberKeyValue;

kv={

}

kv.24="Hello"; // Unexpected keyword or identifier.

kv.32="God is great"; // Unexpected keyword or identifier.

kv.cool=25;

kv.ice=343;

kv.suger = 34;

console.log(kv);

}

eg10();

/\*

\*/

K.23 = value; This is synthetically incorrect

After transpile getting code

function eg10() {

var kv;

kv = {};

kv;

.24 = "Hello"; // Unexpected keyword or identifier.

kv;

.32 = "God is great"; // Unexpected keyword or identifier.

kv.cool = 25;

kv.ice = 343;

kv.suger = 34;

console.log(kv);

}

eg10();

/\*

\*/

Which is incorrect