Typescript handson:

Why we need typescript?

1. In Javascript, type safety is not there. It supports dynamic typing. Example:

*var* i=20;

console.log(i);

*var* i="Raja";

console.log(i);

console.log("WOrking finee.");

But in typescript, it is static typing. Means, once declared then its type is static. Does not change.

In typescript: two.ts

*var* i=20;

*var* i="Raja";

we are not allowed to do like above code. The data type of the variable i cannot be dynamically changed.

1. Typescript is compiled.

This helps catch type-related errors at compile-time, which can prevent many bugs and make the code more robust and reliable.

TypeScript is a programming language and a superset of JavaScript that adds optional static typing to the language. It was created by Microsoft and has gained significant popularity among developers and companies for several reasons:

1. **Type Safety**: TypeScript introduces static typing, allowing developers to specify the types of variables, function parameters, and return values. This helps catch type-related errors at compile-time, which can prevent many bugs and make the code more robust and reliable.
2. **Code Maintainability**: With static typing, TypeScript provides better code readability and maintainability. The type annotations serve as documentation for the codebase, making it easier for developers to understand the intent of the code. Additionally, as projects grow larger and more complex, TypeScript helps in managing the codebase by enforcing clear contracts between functions and components.
3. **Intellisense and Tooling Support**: TypeScript provides excellent tooling support, including intelligent code completion, code navigation, and refactoring tools. This allows developers to be more productive and write code more efficiently, thanks to the code editor's ability to offer suggestions and catch errors in real-time.
4. **Improved Collaboration**: With TypeScript, teams can collaborate more effectively. Since the types are explicitly defined, team members can better understand each other's code, reducing misinterpretations and potential misunderstandings.
5. **Gradual Adoption**: TypeScript is designed to be compatible with existing JavaScript code. This means you can start using TypeScript in a project by incrementally adding type annotations to existing JavaScript code. This gradual adoption path makes it easier for developers and teams to transition to TypeScript without requiring a complete rewrite of their codebase.
6. **Future JavaScript Features**: TypeScript often incorporates upcoming JavaScript features and syntax, even before they are officially supported by browsers. This allows developers to use modern JavaScript features while transpiling the code to older versions of JavaScript that are compatible with all major browsers.
7. **Popular in the Ecosystem**: TypeScript has gained widespread adoption and is widely used in various large-scale projects and libraries. This means there is a rich ecosystem of tools, libraries, and resources available for developers using TypeScript.
8. **Strong Community Support**: TypeScript has a vibrant and active community that contributes to its development, maintains various packages, and provides support through forums and other online platforms.

Overall, TypeScript offers numerous advantages for JavaScript development, making it a powerful tool for building scalable, maintainable, and robust applications, especially in large and complex projects or teams. However, it's essential to consider your project's specific needs and the familiarity of your team with TypeScript before deciding to adopt it.

TypeScript supports the following data types:

1. \*\*Boolean\*\*: Represents a true or false value.

```typescript

let isDone: boolean = false;

```

2. \*\*Number\*\*: Represents both integer and floating-point numbers.

```typescript

let age: number = 30;

let pi: number = 3.14;

```

3. \*\*String\*\*: Represents a sequence of characters (text).

```typescript

let name: string = "John Doe";

```

4. \*\*Array\*\*: Represents a collection of elements of the same type. TypeScript supports arrays of specific types or a union of types.

```typescript

let numbers: number[] = [1, 2, 3, 4, 5];

let fruits: Array<string> = ["apple", "banana", "orange"];

```

5. \*\*Tuple\*\*: Represents an array with a fixed number of elements, where each element can have a different type. The types of elements are known at specific indices.

```typescript

let employee: [string, number] = ["John Doe", 30];

```

6. \*\*Enum\*\*: Represents a set of named constants (enumeration).

```typescript

enum Color {

Red,

Green,

Blue,

}

let favoriteColor: Color = Color.Blue;

```

7. \*\*Any\*\*: Represents a value for which the type is not known or doesn't matter. It allows you to opt-out of type checking for specific variables.

```typescript

let dynamicValue: any = "This could be anything!";

```

8. \*\*Void\*\*: Represents the absence of a value. It is commonly used as the return type for functions that don't return a value.

```typescript

function logMessage(message: string): void {

console.log(message);

}

```

9. \*\*Null and Undefined\*\*: Represents null and undefined values, respectively.

```typescript

let someValue: null = null;

let anotherValue: undefined = undefined;

```

10. \*\*Never\*\*: Represents the type of values that never occur. It's often used to denote functions that throw exceptions or enter infinite loops.

```typescript

function throwError(message: string): never {

throw new Error(message);

}

```

11. \*\*Object\*\*: Represents non-primitive types (anything that is not number, string, boolean, null, or undefined). It is a type that describes any JavaScript object.

```typescript

let person: object = { name: "John Doe", age: 30 };

```

12. \*\*Union Types\*\*: Represents a type that can be one of multiple types. It is denoted by using the pipe (|) symbol.

```typescript

let numberOrString: number | string = 42;

numberOrString = "hello";

```

13. \*\*Intersection Types\*\*: Represents a type that combines multiple types. It is denoted by using the ampersand (&) symbol.

```typescript

type Point2D = { x: number; y: number };

type Point3D = { z: number };

let point: Point2D & Point3D = { x: 1, y: 2, z: 3 };

```

These are the core data types in TypeScript. Additionally, you can create your own custom types using interfaces, type aliases, and classes to represent complex data structures and models. TypeScript's strong type system enhances code reliability, maintainability, and readability, making it a popular choice for many developers and projects.

In typescript, we can declare a variable of multiple data types / types.

int x; is x:number

void sum() is sum():void

Typescript notes

javascript

why we need typescript?

every time, javascript new version comes, the browsers need also to be upgraded

so how to introduce new features in javascript, without the need to upgrade the browsers?

so lets keep javascript same. and introduce a new language, which can contain

adv features. on compilation, that new language code should be translated into

javascript.

browsers cannot execute typescript. they can execute javascript.

var x:string;

var y: number|string;

var z:any;

z="raja";

z=12;

y=12;

y="siva";

y=new Date(); //error because only number or string is allowed

y={"id":12,"name":"raja"}; //error

benefits:

1)compiled

in javascript, (since the code is not compiled), the syntax errors are not visible.

to check the errors, i need to run the code.

in typescript, syntax errors are visible during compilation. so i do not need to run the code.

2) typed

strong typing (variables can be declared as a specific type)

3) OOP

Classes, Interfaces, Inheritance

-------------------------

components of typescript:

1) Language

2) Compiler

3) Language service:

auto completion

signature help

code formatting

-----------------------------

typescript ignores space, tab, white space in a program

typescript is case sensitive. it differentiates lower case with upper case.

semi colon is optional at the end.

x : number (no need for ; because new line space)

y : string

this is fine.

but

x: number y:string

this is not okay

----------------------

comments

single line comments //

multi line comments /\* \*/

----------------------

variables / identifiers (name of anything)

1) cannot start with a digit

2) can contain characters, numbers.

3) symbols are not allowed

except \_ and $

4) identifiers cannot be keyword

5) must be unique (within the scope)

6) case sensitive

7) cannot have space in between

-------------------------------------------

Data types:

number

string

boolean

enum (fixed set of values)

enum Days {Monday, Tuesday, Wednesday};

let x:Days=Days.Monday;

console.log(x); //0

void (functions)

null no value is assigned

undefined no type is assigned

any

var z:any;

z=20;

z='rama';

z=true;

never

function fn1():never

{

//somewhere we raise exception for sure

}

array

in js, var arr=[1,2,3,4,5];

let arr:number[]=[1,2,3,4,5];

tuple

let person:[string,number]=["Ram",100];

many variable of diff types inside tuple

TasK:

write a typescript program to declare variables of

number

string

boolean

enum

any (also declare variable of

2 types.

ex: var x:string|boolean

x="true"; //string

x=true; //boolean

display the variables in console.

--------------------------------------------

var x: any;

x='rama';

console.log(x);

x=20;

console.log(x);

-------------------------

variable declarations:

int x; // c, c++, c#, java, j#

but in typescript

let x:number;

1) declare its type and value in one statement

var x1 : string = "india";

2) declare its type but no value

var x1 : string;

3) declare its value but no type

var x1 : "india";

4) declare variable but no type or no value

var x1;

-----------------------------

Type Assertion in typescript: (change variable from one type to another)

like type casting or conversion

in c language:

int x=(int) f; //f is a float

in typescript:

var str="1"; //based on value, str is string

var xy:number=str; //error

solution:

var xy:number=<number>str; //cannot cast string into number (so error)

var xy:number=<number> <any>str; //correct

console.log(typeof xy);

console.log(typeof(xy));

a+b i know the formula

sum(a,b) i may not know the formula. but still get output

Inferred typing:

var x1:number;

x1='india'; //not allowed because x1 is a number

data type checking in compiler level. (in vs code, underlining while typing)

Variable scope:

Global scope

Class scope

Local scope

a variable declared outside any function/ class (global)

var global=12; //global variable

class Numbers{

num\_val=13; //class variable

static sval=10; //static field

fn1():void

{

var local\_var=14; //local variable

}

}

var obj1:Numbers=new Numbers();

obj1.num\_val=100; //class variable using object

obj1.sval=200; //error (static variable cannot be referred using object)

if i create 5 objects of this class Numbers. each object will have separate copy of class variable.

but only 1 copy of static variable.

correct way to access static variable is:

Numbers.sval=200; //class.static variable

code:

var global\_num = 12 //global variable

class Numbers {

num\_val = 13; //class variable

static sval = 10; //static field

storeNum():void {

var local\_num = 14; //local variable

}

}

console.log("Global num: "+global\_num)

console.log(Numbers.sval) //static variable

var obj = new Numbers();

console.log("Global num: "+obj.num\_val)

-----------------------------

what are all other operators we have?

categories:

Arithmetic operators:

+ - \* /

% ++ --

z=x%y; after diving x by y, the reminder is stored in z

++ increment by 1

-- decrement by 1

Logical operators:

! && ||

Relational operators:

> < >= <= == !=

Bitwise operators:

& | ~ ^ << >> >>>

Assignment operators:

= += -= \*= /= %=

Ternary/conditional operator:

?:

String operator:

+ (this is concatenation)

Type Operators:

typeof

instanceof

------------------------------------------------

Programming constructs:

if

if else

if else if -else

switch case

for loop (definitive loop)

for(;;) (infinite loop)

while(true) (infinite loop)

while (indefinite loop)

do while (indefinite loop)

break

continue

for(var i=0;i<10;i++)

{

console.log(i); //prints 0 to 9

}

var i;

for(i=0;i<10;i++)

{

}

var i=0;

for(;i<10;i++)

{

}

var i=0;

for(;;i++)

{

if(i>=10)

break;

}

var i=0;

for(;;)

{

if(i>=10)

break;

i++;

}

Task:

Each player will be allowed to throw a dice 2 times. The points for each player will be calculated as follows :

The points scored is the absolute difference between 8 and the sum of the 2 values, provided the sum of the 2 values is less than 8.

In all other cases, the point scored is double the absolute difference between 8 and the sum of the 2 values.

Write a program to calculate the points scored by a player.

test case 1:

var input1=5;

var input2=6;

the output should be : 6

explanation:

the sum of 2 values: 11 (5+6)

abs diff between 8 and sum: 11 and 8 = 3

since the sum (11) is not less than 8, the points is double the abs diff = 2 \* 3 =6

test case 2:

var input1=3;

var input2=2;

the output should be : 3

Task:

Run Length IV

Write a program to find the length of the longest running sequence of even numbers in the given array. Also find the starting index of the longest running sequence of even numbers.

Example :

Consider the array ... [ 2, 16, 15, 2, 6, 8, 10, 7, 22, 4 , 6]

The 3 even number sequences in this array are {2, 16}, {2, 6, 8, 10} and {22, 4, 6}.

The longest running even number sequence is {2, 6, 8, 10}

The length of the longest running even number sequence is 4 and the starting index is 3.

Note:

The array indexing starts from 0.

If there are multiple choices, select the choice where the starting index is minimum.

declaration:

var arr=[ 2, 16, 15, 2, 6, 8, 10, 7, 22, 4 , 6];

OUTPUT:

4

3

HINT:

declare variables as required.

even number is a number whose reminder after dividing by 2 is 0

ex: x%2==0 is true, then x is an even number

array's first element starts at index 0

walk through the array from 1st to last. keep variables to find the length and index.

highest of 3 numbers

x=5;

y=3;

z=8;

var max=0;

if x is greater than max, then max is updated.

x is 5 max=5

y is 3 max=5 (not updated)

z is 8 max=8 updated

solution:

var arr=[ 2, 16, 15, 2, 6, 8, 10, 7, 22, 4 , 6];

var count=0;

var maxCount=0;

var startingIndex=-1;

var maxStartingIndex=-1;

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

{

// console.log(arr[i]);

//check if the current element is even or not

if(arr[i]%2==0)

{

count++;

if(count==1)

{

//there is a beginning

startingIndex=i; //i is the current index

}

if(count>maxCount)

{

maxCount=count;

maxStartingIndex=startingIndex;

}

// console.log("Count is: "+count);

// console.log("MaxCount is: "+maxCount);

}else

{

count=0;

}

}

console.log("Max count is "+maxCount);

console.log("Starting index is "+maxStartingIndex);

-----------------------------------------------------------------------------------------------------

Task:

Write a program to find the length of the longest consequtive sequence of any number in the given array. Also find the starting index of the longest such sequence.

Example :

Consider the array ... [ 3, 5, 6, 6, 6, 6, 7, 7, 2, 1, 8]

6 appears 4 times consequtively in this array.

The length of the ongest consequtive sequence of any number in the given array is 4 and the starting index is 2.

Note:

The array indexing starts from 0.

If there are multiple choices, select the choice where the starting index is minimum.

count>maxCount even if another seq have same length as maxCount, it is ignored.

count>=maxCount if another seq is found, starting index is updated;

Task:

Write a program to find the product of all 2 digit numbers in an array.

var arr=[11,2,3,10,5,8,12];

<100 ????? did you check if number is <100

----------------------------------------------

Functions: (in typescript)

1) defining functions

2) calling functions

3) returning functions

4) parameterized functions

default parameters fn1(x:string='india')

optional parameters fn1(x?:string)

rest parameters (param array) ... ellipses is used for REST parameter (means, varargs)

5) anonymous functions

(anonymous = no name)

like function pointers

we store the function in a variable

var str=function()

{

return "Hello world";

};

//to call this

console.log(str());

6) lambda functions (arrow function)

var str1=()=>{ return "Hello Bharath!"; };

console.log(str1());

Function overloading

suppose there is a function with param array REST parameter

function fn2(a:number, ...x:number[])

{

}

fn2(2,3,4,5,6,7) a=>2 x=> 3,4,5,6,7

fn2(2,3);

a=2

x will take 3

this is correct

fn2(2);

a=2

x will be a 0 sized array

this is also correct

1) Rest parameter (param array) must be the last arg in the list

2) usually, any parameter can be optional. But when there is a rest parameter, there cannot be a parameter which is optional.

when there is a rest parameter in a function, there cannot be any other optional parameters in the function. Rest parameter itself is optional.

what is optional parameter?

{

console.log(x+y);

}

function add(a: number, b: number, c?: number): number {

console.log("Addition of two numbers is ")

return a + b;

}

console.log("Demonstration of optional parameter in Typescript")

let res = add(2,3)

console.log(res)

//let res1 = add(3)

//console.log(res1)

optional parameters must be pushed to the end. After optional parameters, there cannot be a mandatory/required parameter

fn10(2);

fn10(2,3);

the above is not possible in typescript like other languages.

//overloading: defining multiple functions with same name but different parameters. each version of functions have different body. But

// in typescript, same function body for different parameter types.

//templates in c++? <T> generic type.

//same function name. but different behavior. (different function body)

//generic template means, same behavior for different parameters.

//Overloading in typescript

------------------------------------------------------

Task:

Sam is a cricket freak, who loves to collect statistics about the matches. Suppose, a batsman scored X runs which included Y fours and Z sixes, Sam wants to calculate the percentage of the total score he made by running between the wickets.

Write a program to help Sam .

var x=110; //x is the runs scored

var y=3; //y is the number of 4s

var z=8; //z is the number of 6s

output should be: 45.45

var x=60; //x is the runs scored

var y=2; //y is the number of 4s

var z=1; //z is the number of 6s

output should be: 76.66

----------------------------------------------------

Task:

Playing with arrays has become a hobby for Arun. This time Arun wants to find the greatest number in the array such that it is the product of any two numbers in the array.

Write a program to help Arun to the do the task. If no such number exists, then print -1.

[Note: The two number that are used to find the product can be the number itself. For example, if the array elements are [1,2,3] , then 1\*3 = 3 will be the solution for this array.]

var arr=[40,35,30,7,6,5]; //35 being largest num is a coincidence

output: 35

var arr=[2,4,6,89,78];

output: -1

var arr=[-10, -5, 50, 2, 110];

output: 50

for(var i=0;i<4;i++)

{

for(var j=0;j<4;j++)

{

i,j

}

}

0,0

0,1

0,2

0,3

1,0

1,1

1,2

1,3

2,0

2,1

2,2

2,3

3,0

3,1

3,2

3,3

education what are the vowel found in this sentence?

euaio

35,30,7,6,5

one more test case:

-10, -5, 50, 2, 11

to find the product of any 2 numbers that give largest value. people sorted the array (smart). and added last 2 numbers.

some people sort this array. and deal only from last digits

how many of you know Comparable in Java?

a-b return negative number means, which is bigger? b is bigger

a-b return positive number means, a is bigger

a-b return 0 means, a and b are equal

so to compare 2 objects, they need this method.

-------------

To sort array in typescript:

-----------

// function jag(a,b)

// {

// return a-b;

// }

// var arr:number[]=[11,1,20,6,8,3];

// arr.sort(jag);

// console.log(arr);

var arr:number[]=[11,1,20,6,8,3];

arr.sort((a,b)=>b-a); //a-b is asc. b-a is desc.

console.log(arr);

-----------------------

Class is a blueprint for creating objects

Object is an instance of Class

Inheritance

Overriding

Interface

if in E-R diagram, an Entity is translated into a Table in Back End,

the same Entity is translated into Class in front End

class is a Type (user defined type)

how to create a class named as Employee

class Employee

{

}

A class can contain:

properties

methods

class Employee

{

employeeId:number;

firstName:string;

lastName:string;

}

how to create an object of a class.

object

class Employee

{

employeeId:number;

firstName:string;

lastName:string;

}

var rama:Employee=new Employee();

rama.employeeId=1;

rama.firstName="Rama";

rama.lastName="Krishna";

console.log(rama);

----------------------

constructors:

when no constructors are defined in a class, the compiler creates one. (default constructor).

default constructors have no arguments (no parameters)

class Employee

{

employeeId:number;

firstName:string;

lastName:string;

static companyName:string;

constructor(id:number,fname:string, lname:string)

{

this.employeeId=id;

this.firstName=fname;

this.lastName=lname;

Employee.companyName='Mphasis';

}

}

var rama:Employee=new Employee(12,"Jag","India");

// rama.employeeId=1;

// rama.firstName="Rama";

// rama.lastName="Krishna";

Employee.companyName='MPhasis';

console.log(rama);

-----------------------

class Person

{

name:string;

age:number;

private marks:number;

constructor(nm:string, ag:number)

{

this.name=nm;

this.age=ag;

}

}

class Student extends Person{

subject:string;

constructor(nm:string, ag:number, sb:string)

{

super(nm,ag);

this.subject=sb;

}

}

var rama:Student=new Student("Raja",22,"Maths");

// rama.marks=90; //not allowed because marks is private in Person

// rama.subject="typescript"; //inheritance is for entensibility

console.log(rama);

-----------------

class Person

{

name:string;

age:number;

private marks:number;

constructor(nm:string, ag:number)

{

this.name=nm;

this.age=ag;

}

display()

{

console.log(this.name);

console.log(this.age);

}

}

class Student extends Person{

firstName:string;

lastName:string;

subject:string;

constructor(fn:string,ln:string, ag:number, sb:string)

{

super(fn+' '+ln,ag); //super class constructor must be called in the first line only

this.subject=sb;

// super.name=fn+' '+ln;

}

display() //overriding. because, display() already exists in super class

{

super.display();

console.log(this.subject);

}

}

var rama:Student=new Student("Raja", "siva",22,"Maths");

// rama.marks=90; //not allowed because marks is private in Person

// rama.subject="typescript"; //inheritance is for entensibility

// console.log(rama);

rama.display();

----------------------------------------------

Interfaces

interface Person

{

speak();

}

class Student implements Person

{

speak()

{

console.log("Student speaks");

}

}

class Teacher implements Person

{

speak()

{

console.log("Teacher speaks");

}

}

var rama:Student=new Student();

var siva:Teacher=new Teacher();

function talk(x:Person)

{

x.speak(); //x may be a Student or Teacher or any one who implements Person

}

talk(rama);

talk(siva);

class Webex

{

presenter:Person; //Student/Teacher anyone can present

}

--------------------------

Interfaces separates the declaration of functions from its implementation.

Rest API will return objects

object means, there should be a Class.

To declare object here, we do not have the class.

They will not share the class with us. (containing original logic)

So they can share the interface only with us. Relate it to the broucher and book example

-----------------------------------

Activity:

What are namespaces in typescript? Why they are required? how to use namespaces?

typescript 4.0.3 or earlier

1) create jag1.ts

export class Student

{

}

2) create jag2.ts

import {Student} from './jag1'

var rama:Student;