**TypeScript**

**Setup Of TypeScript**

**Title: Variables**

Syntax: To declare the variables.

Scope-name variable-name:data-type;

const num1:number = 10;

data-type inference: To deduce the datatype of the variable by the value assigned to it.

let myName = "mridul"; // data-type inference by TS

**Drawbacks of JavaScript?**

==============================

  - It is a dynamic language

  - Forecast the errors that might occur during the execution

  - Errors may occur at right time due to wrong data type

Example:

     var x = 10;

      console.log("x =   ", x); // 10

      console.log("Data type of x - ", typeof x); // number

      // performing operation on the variable

      var y = x.toUpperCase(); // typeof x is number and we are perform string operation

      console.log("y = ", y); //

Web Development

                                                    HOME              ABOUTUS           PRODUCT

REQUIREMENT - Reliable code in terms of data type error

Solution : Use Language that provides data types for WEB applications (UI)

=========================================

 TYPESCRIPT , DART, ELM ….- Languages (Object-Oriented Language)

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

 Java Script (scopes, control statements, functions, objects..)

===================================================

What?

TypeScript Language based on JavaScript with extended features

Extended features (class, data types, interfaces, modules…)

TypeScript -------------> statically typed language, check types at compilation time.

Data types at language level

         string

         number

         boolean

Install typescript

================

  - Compiler (tsc)

Compilation process of Typescript

==================================

Source Code                                                               Compiler                       Interpreted

\*.ts ---------------------------------------------------------------------->         \*.js  -----------------------> JS Engine  in Browser

Features of TypeScript

===========================

1. It is superset of JavaScript
2. It is an object-oriented language. However, not necessary that \*.ts starts with class

 function(){…}  class{….},    interface {…}

1. It is written by Microsoft
2. Typescript syntax is similar to C#, Java
3. X = 10
4. // X="john" // wrong assignment
5. <https://www.typescriptlang.org/>
6. TypeScript code converts to JavaScript, which **runs anywhere JavaScript runs**: In a browser, on Node.js
7. It provides data types. Hence, performs data type checking at the compile.

Typescript                                                                                                                     JavaScript

class Employee{                                                                                 function Employee(){

                                             ======================>

}                                                                                                              }

             string name="john";        =======================>                       name = name \* 100; // error runtime

           name.getEmployeeDetails()  //error  at compile-time

 // name = name \* 100; // error at compile-time

(MEAN) -> MongoDB, Express FRAMEWROK   + NODE API + ANGULAR (UI)  + NODE API (POSTGRES API)

NODEJS = JS Engine + NODE API (DB API, NETWORK API, OS API ) + NPM (NODE PACKAGE MANAGER)

NODEJS API - FUNCTIONS, CLASS, OBJECTS…

What is the runtime platform from NODEJS?

   JS Engine -------------> \*.js file for execution

Command to run js file on nodejs platform : node <<file-name>>  // by default it takes .js file.

test.js

Cmd: > node test

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

Java - language  with features (data type, control statement, API - lang package, io package,…, Arrays, collection, JDBC API)

As platform , Java ----------> JRE (Java Runtime Env.)/JVM (Java Virtual Machine)

 \*.class  ------------> JRE ----------->Execute the class file.

* Account.class
* Cmd : java Account

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

Development Styles

====================================

     Traditional Web application

                 project

                   src

                          - html

                                             \*.html

                          - css

                                               \*.css

                         - js

                                                \*.js

ADD BOOTSRAP in the existing project

====================================

1. Download bootstrap. Unzip it and include bootstrap.min.css & bootstrap.min.js files in the HTML page

   B.    Add the cdn link for .css & .js file

          <link href="https://cdn.jsdelivr.net/npm/bootstrap@4.3.1/dist/css/bootstrap.min.css"/>

Steps:

            1. Download Bootstrap in the project folder

            2. Unzip the downloaded folder

            3. create index.html and add these two lines in the <head> section

<head>

      <link  rel="stylesheet" href="./bootstrap-4.3.1-dist/css/bootstrap.min.css"/>

     <script src="./bootstrap-4.3.1-dist/js/bootstrap.min.js" > </script>

    <title> My Bootstrap Manual Adding </title>

</head>

     4.  Apply styles from bootstrap

          <button class="btn btn-lg btn-primary"> Click to Submit </button>

   5. Right-click index.html and open in browser.

Typescript syntax to declare variable

var variable-name : datatype = value;

====================================================================

NODEJS Installation

                 - NPM INSTALLED     along with NODEJS

You can check

  CMD:   node --version  (V14.x)

               npm --version   (6.x)        is a package manager tool   for JS (JS library - jquery, express,react, CSS library - Bootstrap, SASS, js framework - angular,vue, knockout

NPM is a repository for JS libraries and frameworks

you can use NPM to download JS libraries in your application.

Features of NPM

==========================

1. NPM is a repository for JS libraries and frameworks
2. Command line tool (CLI) - runs on CMD as commands
3. Commands:
   1. Install library / uninstall library
   2. Install with npm - it can be global or local
   3. C:\myreactproj> npm install  <<js-library>>@version   
      This will install library in the local project / folder.

What is package.json?

  - records metadata (information) about the project when done by npm

 - record name of libraries with their version downloaded  by npm

* Syntax to create package.json file , CMD: projectfolder> npm init

Give answers to the questions by pressing enter.

Demonstration

==============================================================

1. Create a new folder named TypeScriptCodes
2. Open the newly created folder in VSCode
3. Also, open the same folder in the CMD
   1. D:\projects\.....\WebDevelopment\TypeScriptCodes>
4. Commands in CMD:
   1. generate package.json
      1. npm init
5. To install bootstrap library using npm - npm install bootstrap@4.3.1
   1. Download bootstrap library files in the folder node\_modules inside the project folder
   2. Verify package.json to view the entry of bootstrap dependency
   3. Verify node\_modules/bootstrap/dist to view CSS & JS folder
   4. Create index.html and add the following code

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>PROJECT USING NPM</title>

    <link

      rel="stylesheet"

      href="./node\_modules/bootstrap/dist/css/bootstrap.min.css"

    />

    <script src="./node\_modules/bootstrap/dist/js/bootstrap.min.js"></script>

  </head>

  <body>

    <button class="btn btn-info">Login</button>

</body>

</html>

* 1. Right-click the index.html file and open in live server

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

To install typescript using npm

1. CMD : project-folder> npm install typescript
   1. By default, latest version download and installed in our project
   2. You can verify : package.json file & node\_modules folder to view Typescript entry & installation
   3. Create src folder in the project
   4. Add variables.ts file in the src folder
   5. Write the shared code

// declare a variable : syntax - <scope> variable-name : data-type = value;

var num1 : number = 10; // readability of code

// compute logic

var sum = 0; // data-type inference by TS

//sum = "john";

var y: string;

 y = sum + "john";

var flag: boolean = true ;

console.log ("num1 = " , num1);

console.log("y = " , y);

console.log("flag = ", flag);

* 1. Compile the .ts file to .js file using command

…..node\_modules/typescript/bin>tsc

…..node\_modules/typescript/bin>tsc variables.ts

If not compile errors, file will be compiled to .js file

* 1. Add the .js file in the <script src="./src/variables.js"></script> in the html page.

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>PROJECT USING NPM</title>

    <link

      rel="stylesheet"

      href="./node\_modules/bootstrap/dist/css/bootstrap.min.css"

    />

    <script src="./node\_modules/bootstrap/dist/js/bootstrap.min.js"></script>

  </head>

  <body>

    <button class="btn btn-info">Login</button>

    <script src="./src/variables.js"></script>

  </body>

</html>

* 1. Right-click index.html and open in Live Server. Press F12 to open DevTools and check the output.

Project Structure of Typescript application

======================================

       project

            -  package.json             // entries for dependencies in the project, npm init

            -  node\_modules (folder)

                        package file (s)

            - src

                    \*.ts

            -index.html

            -index.css

NPM - Node Package Manager  - repository of JS libraries

To install any library : npm install <package-name/library-name>@version

To compile ts code to js

node\_modules\typescript\bin> tsc   <file-name>.ts   <file-name>.ts

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

Compiler converts code into machine language (11100001111)

TS Compiler convert code into JavaScript code, hence  it is called a TypeScript TRANSPILER.

Transpilers, or source-to-source compilers, are tools that read the sourcecode written in one programming language and produce the equivalent code in another programming language

From <<https://www.google.com/search?q=typescript+transpiler&rlz=1C1GCEU_enIN973IN973&oq=typescript+tra&aqs=chrome.1.69i57j0i512l9.11396j0j15&sourceid=chrome&ie=UTF-8>>

Any datatype:

Any is the default data-type in typescript if not specify data type.

The type check will not done and it will be skipped.

function generateRandomNumber(val:any){

    return Math.round(val);

}

***Functions in TypeScript***

Function function-name(parameter: datatype, … ) : return data-type{

Return;}

**Declaring Object Datatypes in TS**

Incorrect way

Const person: object = {

Name: “Mridul”;}

Person.name;

Here you cannot access the name property of the person object because datatype is defined as object but what properties exist in it are not defined so typescript does not identifies the property.

Correct way

const person: {

    name: string,

    age: number

    }

    = {

    name: "mridul",

    age: 30

};

correct way 2:

Here it will do type inference.

const person = {

    name: "mridul",

    age: 30

};

**Defining the arrays in TS**

Correct way:

let names: string[] = [];

names[0] = "dinseh";

incorrect way: It will throw error that names[0] is undefined as js will consider it as variable

let names: string[];

names[0] = "dinseh";

correct way

let names: string[] = [];

// names[0] = "dinseh";

names = ['mridul', 'rajbhar'];

**Defining Tuple Data type**

Syntax:

Scope-name variable-name: [datatype, datatype, ….]

e.g

let myTuple : [number, string];

TS will consider it as the tuple but JS will consider it as array.

Here the array length will be fixed and datatype for the particular element will alo be fixed.

Correct way:

myTuple = [3, "mirdul"];

incorrect way:

in declaration first element is number and second is string but here

first is string and second element is number so it is incorrect.

myTuple = ["mrisul", 3];

incorrect way:

in definition the length is 2 but here the length is 3 so incorrect way.

myTuple = [3,"mridul", 2];

**Enum type in TypeScript**

It is Set of constants and it is iterable.

Correct way:

enum Role {

    ADMIN = "myAdd",

    READ\_ONLY = 7,

    AUTHOR = 100

};

Correct way:

Here by default Admin will be assigned value 0 and next constant will have value:

Previous\_constant\_value + 1;

enum Role {

    ADMIN ,

    READ\_ONLY ,

    AUTHOR

};

**TypeScirpt Any type:**

There is no type checking you can assign data of any type to the variable.

**TypeScript Union Type (|) 🡪 Pipe symbol:**

Here input1 and input2 both the parameters are taking number as well as string as the input using ‘|’ pipe operator in the function.

function combine(input1: number | string, input2: number | string){

    if(typeof input1 === 'number' && typeof input2 === 'number')

        return input1 + input2;

    else

        return input1.toString() + input2.toString();

}

const combineNum = combine(12, 45);

const combineStr = combine("Mridul ", "Rajbhar");

console.log(combineNum);

console.log(combineStr);

**TypeScript Literal Types:**

Literal types are used when we want variables to have only certain values nothing other that it.

Eg

function combine(input1: number | string, input2: number | string, result: 'num' | 'str'){

    if(typeof input1 === 'number' && typeof input2 === 'number')

        return input1 + input2;

    else

        return input1.toString() + input2.toString();

}

const combineNum = combine(12, 45, 'num');

const combineNum = combine(12, 45, 'number'); //error

Here parameter, result can have only two values “num” or “str”

In second function call we are passing ‘number’ which is not allowed.

**Type Alias**

Type type-name = data-type | data-type ….

type comninable = number | string;

type conversionDescriptor = 'num' | 'str';

function combine(input1: comninable, input2: comninable, result:conversionDescriptor ){

    if(typeof input1 === 'number' && typeof input2 === 'number')

        return input1 + input2;

    else

        return input1.toString() + input2.toString();

}

Type aliases can be used to "create" your own types. You're not limited to storing union types though - you can also provide an alias to a (possibly complex) object type.

Example 1:

1. type User = { name: string; age: number };
2. const u1: User = { name: 'Max', age: 30 }; // this works!

Example 2

1. function greet(user: { name: string; age: number }) {
2. console.log('Hi, I am ' + user.name);
3. }
5. function isOlder(user: { name: string; age: number }, checkAge: number) {
6. return checkAge > user.age;
7. }

To:

1. type User = { name: string; age: number };
3. function greet(user: User) {
4. console.log('Hi, I am ' + user.name);
5. }
7. function isOlder(user: User, checkAge: number) {
8. return checkAge > user.age;
9. }

**Using Function as a type:**

Here variable will accept the function with no parameters and return type number.

let combineValues: ()=>number;

Here variable will accept the function with two number parameters and return type number.

let combineValues: (a:number,b:number)=>number;

**Function Types And Callbacks**

function addAndHandle(n1:number, n2:number, cb:(num:number)=>void){

        const result = n1 + n2;

        cb(result);

    }

addAndHandle(50, 20, (result) => {

    console.log(result);

});

“Unknown Type”

**“Unknown” types also take any type of data in the variable but you cannot assign the value of unknown type to the variable without type-checking but “any” type data can be assigned to variable without the “type-checking”.**

let userInput1: unknown;

let userName: string;

let userInput2: any;

userInput1 = 5;

userInput1 = "Mridul";

userName = userInput1; // “error: cannot assign”

userInput2 = 10;

userName = userInput2; “no type checking”

userInput2 = "Rajbhar";

**“Never Type”**

**When we are sure do not need to return any type of the value, not even null or undefined.**

**Note: Void will return undefined or null value.**

“Spread Operator”

**It will pull out all the elements from the objects and arrays.**

**Example 1:**

const hobbies = ['Sports', 'Cooking']

const activeHobbies = ['Hiking']

activeHobbies.push(...hobbies);

console.log(activeHobbies);

**Example 2:**

const obj = {name: "Mtidul", age: 22}

const keyVal = {...obj};

console.log(keyVal);

**“Rest Parameters”**

**When we expect the list of values we use the rest parameters.**

**Example 1:**

**Here in the function addRest we have parameter `…numbers` means it will take variable number of argumnets.**

const addRest = (...numbers: number[]) => {

    return numbers.reduce((curResult, curValue)=>{

        return curResult + curValue;

    }, 0);

}

console.log(addRest(12,34,22/7,65));

**Exmaple 1:**

const hobbies = ['Sports', 'Cooking']

const activeHobbies = ['Hiking', 'Drawing']

activeHobbies.push(...hobbies);

const [hobby1, hobby2, ...remainingHobbies] = activeHobbies;

**output**

**hobby1: Hiking**

**hobby2: Drawing**

**remainingHobbies: [ 'Sports', 'Cooking' ]**

**“Pulling elements from objects”**

**OOPS Concept**

**“Class”**

**Classes are blueprints of the objects. They define properties and methods of the function.**

**“Objects”**

**Objects are instances of the classes i.e based on the classes.**

**By default variables in class are public.**

**Var and let scopes are not used to declare the class variables, but it can be used within the function of the class.**

**“this”**

**this refers to the current object.**

**“Constructor”**

**Method/ or function within the class.**

**Initialize the property of the object during object creation.**

**You can only create one object in the constructor.**

**“private”**

**Private properties are not accessible outside the class.**

**“public”**

**Public properties are accessible outside the class.**

**“readonly”**

**Once you apply “readonly” property to any variable then you cannot change its value after initialization.**

**“Inheritance”**

**Inheritance is used when we have a IS-A type relationship.**

**For child class to inherit properties we use “Extends” keyword.**

**e.g**

**class ChildClass extends ParentClass{}**

**When instantiating the child object you have to call base class constructor first using super() method within the child class constructor and also mention the parameter in super().**

**e.g**

**class ChildClass extends ParentClass**

**{**

**constructor(parentParameter1, parentParameter2, childParameter1){**

**super(parentParameter1, parentParameter2); 🡪 will call parent class constructor.**

**}**

**}**

**Overrinding is the supported in the TypeScript.**

**Multiple Inheritance is not supported in the TypeScript.**

**“protected”**

**Protected members of the class can be accessed only within the class and by the child class other than that it cannot be accessed.**

**“get”**

**To access/update the private variables of the class we need “get” and “set” methods.**

**TS provides “get” and “set” keywords to write getter/access and setter/update methods.**

**Example:**

    public get holderName(){

        return this.\_holderName;

    }

    public set holderName(holderName: string){

        this.\_holderName = holderName;

    }

account2.balance = 25000;

console.log(account2.balance);

var account2 = new Account(); //account2 is the reference variables pointing to object

console.log(account2.accountId); // “undefined”

**Here the account2 is the ”reference variables” pointing to the object i.e variables are storing the address of the objects means that variables are pointing to the objects.**

**Objects in the TS are of reference types.**

**State of an object is represented when memory is allocated for the object.**

**By default the value of the variables in the object is “undefined”.**

**“Static Properties And Methods”**

**Static properties and methods are accessed directly by the class name and we don’t need to create the instance of those class i.e for static properties and methods only one copy exists.**

**Static properties and methods cannot be accessed by the instances only by the classname.**

**Constructors cannot be marked as static.**

**“abstract”**

**Abstract classes are mainly for inheritance where other classes may derive from them.**

**We cannot create an instance of an abstract class.**

**An abstract class typically includes one or more abstract methods or property declarations.**

**The class which extends the abstract class must define all the abstract methods.**

**The base class defining the abstract functions must not have implementations.**

**“Singleton Design Patterns”**

**Singleton design pattern is used when we have to make sure that oinly one instance of the class is created.**

**To do that we have to create the private constructor and a static method and property in the class.**

class AccountingDepartment extends Department{

    private static instance: AccountingDepartment;

    private constructor(\_id: string, private reports:string[]){

        super(\_id, 'Accounting');

    }

    static getInstance(){

        if(AccountingDepartment.instance)

            return this.instance;

        this.instance = new AccountingDepartment("ACC\_1001", []);

    }

}

**“Interfaces”**

**Interfaces are just used to define the structure without writing the concrete value.**

**Class that implement the interface has to implement all the methods and properties.**

**We can add readonly property in the interface.**

**Interfaces can also be extended.**

**e.g**

interface Named{

    readonly name: string;

}

interface Greetable extends Named{

    greet(phrase: string): void;

}

**The class which will implement the Greetable interface has to implement properties and functions of Named interface also.**

**Interface can also be used as custom types.**

interface AddFn {

    (a: number, b:number): number;

}

let add: AddFn;

add = (n1:number, n2:number) => {

    return n1 + n2;

}

**“Optional Property”**

**If we want that property must be optional means if we want to use it than we can or else not. We use `?:` to define optional property.**

**Name ?: string 🡪 This is optional property.**

**We can have optional property in interface and constructor too.**

**“Module”**

**By default all the TS files are considered as modules.**

**“export” 🡪 to make any files available**

**“import” 🡪 to load a particular file**

**“Generic Types”**

**The component that can be used with variety of the datatypes and not just by single type is generic type component.**

**Array Generic Types: Array<T>**

**Syntax:**

const names:Array<string|number> = [];

**“Creating Own Generic Types”**

function merge<T,U>(objA: T , objB: U ){

    return Object.assign(objA, objB);

}

const mergedObj = merge({name: "Mridul"}, {age: 30});

**Here T and U are considered as the types and they can take any type of the argument, different name ensures that they can or cannot be same.**

**Example 2:**

function merge<T extends object,U extends object>(objA: T, objB: U ){

    return Object.assign(objA, objB);

}

const mergedObj = merge({name: "Mridul"}, 30);

**Above code will show error as we are passing 30 which is number to parameter U which extends as object.**

**Here we use extends keyword to make sure that argument s always object even if it may have different structure.**

**Example 3:**

interface Lengthy{

    length: number;

}

function countAndDescribe<T extends Lengthy>(element: T){}

**extends Lengthy will make sure all elements passed are having Lengthy property.**

**“keyof” operator**