TypeScript

- TypeScript is a programming language.
- TypeScript used to build the Angular 2+(Angular 2 to Angular 9) and React application.
- TypeScript introduced by MicroSoft.
- TypeScript is the object oriented programming language.
- SuperSet Of javaScript also called as TypeScript.
- "ES6" standards and special tools also called as TypeScript.
- Web Technologies(HTML,HTML5)won't understands TypeScript.
- Web Technologies understands only JavaScript.
- So,As a TypeScript developer we must convert TypeScript to Equalent JavaScript.
- Converting TypeScript to Equalent javaScript is called as "Transpilation".
- "tsc" is the tool given by TypeScript used to perform the "*Transpilation*".
- "tsc" stands for the "TypeScript Compiler".

Environmental Setup for TypeScript application::

1)download and install NodeJS

- 'NodeJS' is the server side scripting language.
- 'NodeJS' used to develop the servers.
  httpServer,TCP server
- 'TypeScript' installation depending on Node Server.
  - ✓ website:http://nodejs.org/en/download/
    - file:node-v12.13.1-x64.msi

2)download and install git

- 'Git' is the tool used to interact with 'GitHub'.
- 'GitHub' is the opensource repository maintained by MicroSoft.

- "GitHub" repository containes so many open source libraries related to type scripting
  - ✓ website:http://git-scn.com/download/with
    - file:git-2.24.0.2-64-bit.exe

3) install type scripting

- we well install "type script" by using command prompt.
- we will install "type script" by using following command.

  >npm install -g type script@latest
- "npm" stands for node packaging manager.
- "npm"is the integrated tool of node js.
- "g" stands for global installation

4) install visual studio code" is the IDE given by microsoft.

- "visual studio code" IDE is the open source IDE.
- "visual Studio Code" IDE Recomended for UI Application.
- java script,Ttype Script,Angular,React,Mean Stack,MERN Stack

Transpilation:-

Converting Type Script to Equalent JavaScript is called as Transpilation.

- Type Script file have the ".ts" extention.
- "tsc" tool used to perform the Transpilation.

Ex.-

```
>tsc demo.ts

>node demo.js
```

Note: "node" is the tool,used to execute the Javascript files.

Variables:

- Variable are use to store the data.
- By using variable we store any type of data.

- ❖ We will define variables by using 'var','let' and 'const' keyword.
- ❖ 'let' and 'const' introduced in 'ES6'.

Number Datatype:

- ➢ we can have 5 types of numbers in TypeScript.
- ➢ @decimal @double @hexadecimal @octal @binary
- ➢ hexadecimal numbers should start with "0x" as prefix.
- ➢ octal numbers should start with "0o" as prefix.
- ➢ binary numbers should start with "0b" as prefix.

Syntax: file name:demo.ts(like that)

```
var variablename:datatype = value;

var decimal:number = 100;

var double:number = 100.12345;

var hexadecimal:number = 0xABC123;

var octal:number = 0o123;

var binary:number = 0b1010;

console.log(decimal,double,hexadecimal,octal,binary);
```

Run this command:::

```
>tsc demo.ts

>node demo.js
```

string:

- ✓ we can define strings in 3 ways
- ✓ "", ' ', ``(backtick) [Esc down click]
- ✓ ``(backtick) operator introduced in ES6.
- ✓ ``(backtick) used to define the multiline strings.

```
var sub_one:string = "Angular8";

var sub_two:string = 'NodeJS';

var sub_three:string = `MongoDB`;

console.log( sub_one,

        sub_two,

        sub_three );
```

```
>tsc demo.ts

>node demo.js
```

Ex.1

```
var tbl_name:string = "employees";

var sal:number = 50000;

var sqlQuery:string = `select * from ${tbl_name} where esal>${sal}`;

console.log(sqlQuery);
```

Ex.2

```
var uname:string = "admin";

var upwd:string = "admin";

var tbl:string = "login_details";
```

```
var query = `select * from ${tbl} where uname='${uname}' and
upwd='${upwd}'`;

console.log(query);
```

---

boolean datatype:

```
var flag:boolean = true;

console.log(flag);
```

"any" datatype:

-"any" datatype also called as super datatype

-"any" datatype used to store the all categories of data.

```
var response:any = "server data soon....!";

console.log(response);
```

“array” datatype:

number array:

```
var num_ary1:number[] = [1,2,3,4,5];

var num_ary2:Array<number> = [6,7,8,9,10];
```

forEach():

- forEach() loop introduced in ES6

```
num_ary1.forEach((element,index)=>{

    console.log(element,num_ary2[index]);  });
```

string array:

```
var str_ary1:string[] = [`Angular`,
                         `React`,
                         `NodeJS`,
                         `VueJS`,
                         `React Native`];
var str_ary2:Array<string> = [`MySQL`,
                              `MongoDB`,
                              `SQLServer`,
                              `CouchDB`,
                              `CassandraDB`];
str_ary1.forEach((element,index)=>{
    console.log(element,str_ary2[index]);
});
```

let keyword:

- let keyword introduce in ES6.
- "let" keyword use to define the variable.

Ex.-

```
    var data:number=100;
     console.log(data);  //100
```

```
{
   var data:number=200;
    console.log(data);//200
}
   console.log(data);  //200
```

-In above the example as a developer we are expecting output as 100,but we got 200 instead of 100.

- the above issue technically called as "**global poluting issue**".
- If block code effecting global members technically called as global members technically called as "global poluting issue".
- We can overcome "global poluting issue" by using ***let*** keyword.

Scope rule brack by var keyword:

- In below example we are expecting out as error.but we got 10 as output instead of "Error".

```
for(var i:number=0;i<10;i++){
 };
  console.log(i);//100
```

Solution:

```
for(let j:number=0;j<10;j++){
  }
console.log(j);    //Error:can't find name 'j'
```

NOTE: Because of var keyword ,variable hoistring issue raised.

- Assigning uderfined by JavaScript instead of Error called as variable hoistring.
- In below example we are expected error.
- But we got "undefined" instead of error.

```
console.log(test);

var test:number=1000;
```

Solution:

```
console.log(test1);// Error:Block-scope variable 'test1' used before its declaration.

let test1:number=1200;
```

Duplicate variable allowed by var keyword:

```
var var_one:number=100;

var var_one:number=200;

console.log(var_one);//200
```

Solution:

```
let var_two:number=100;

let var_two:number=200;

console.log(var_two);  //Error:Can't redeclare block-scope variable 'var_two'.
```

Behaviour of 'var' and 'let' is same in function call.

Ex:

```
function myFun(){
    var hello:number=100;
    let wish:number=200;
  console.log(hello,wish); //100 200
    };
   myFun();
```

Var	let
a)var keyword introduce in ES1.	a)let keyword introduce in ES6.
b)Duplicate variables are allowed by var.	b)Duplicate variables are not allowed by let koyword.
c)Global poluting Issue raised by var keyword.	c)we can overcome global poluting issue by using let keyword.
d)variable hoistring raised because of var keyword.	d)we can overcome variable hoistring issue by using let keyword.
e)scope rule break by var keyword.	e)scope rule obey by let keyword.
f)"var" member are global number.	f)'let'member one block scope members.

const::

const keyword also introduced in ES6.reassignment not poossiable for const members.

```
const testVar:string ="Angular8";        //in TypeScript sring 's' is small letter.
  console.log(testVar);  //Angular8
   testvar="Angular9" //Error:Can't assign to testVar because it it const.
```

JSON:

- JSON Stands for Java Script Object Notation.
- JSON used to transfer the data over the Network.
- JSON is ligth weigth Compared to XML.
- Parsing(Reading) of JSON Eazy Compared to XML.
- JSON is Network Friendly Format

Syntax.-

Objects ---- {}

Arrays ---- [ ]

data ---- key & value

key & value separated by using "**:**"

key & value pairs separated by using "**,**"

demo.html(file name)

```
<!DOCTYPE html>
<html>
   <script>
      let obj = {
         sub_one : "Angular",
         sub_two : "NodeJS",
         sub_three : "MongoDB"
      };
      console.log( obj.sub_one,
                   obj.sub_two,
```

```
            obj.sub_three );  //Angular NodeJS MongoDB
        //short cut  (ALT + B)
        // ctrl + shift + i
    </script>
</html>
```

Ex-2

```
<!DOCTYPE html>
<html>
    <script>
        let obj = {
            p_id : 111,
            p_name : "p_one",
            p_cost : 10000
        };

        //for....in
        //for...in loop used to iterate the JSON Objects
        for(let key in obj){
            console.log( obj[key] );
        }
    </script>
</html>
```

Ex-3

```
<!DOCTYPE html>
<html>
    <script>
        let subs = {
                        f_end:{
                            value : "Angular",
                            server:{
                                value : "NodeJS",
                                b_end:{
                                    value : "MongoDB"
                                }
                            }
                        }
        };
        console.log( subs.f_end.value );
        console.log( subs.f_end.server.value );
        console.log( subs.f_end.server.b_end.value );
    </script>
</html>
```

Ex-4

```
<!DOCTYPE html>
<html>
    <script>
        let db = {
            mysql  :  mysql(),
            sqlserver : sqlserver()
        };
        function mysql(){
            return "mysql connection soon...!";
        };
        function sqlserver(){
            return "sqlserver connection soon...!";
        };
        console.log( db.mysql, db.sqlserver );
    </script>
</html>
```

Ex-5

```
<!DOCTYPE html>
<html>
    <script>
        let obj = {
```

```
        fun_one : fun_one
    };
    function fun_one(){
        return "welcome";
    };
    console.log( obj.fun_one() );
  </script>
</html>
```

Ex-6

```
<!DOCTYPE html>
<html>
  <script>
    let modules = {
        login : login,
        logout : logout,
        setCredentials : setCredentials,
        clearCredentials : clearCredentials
    };
    function login(){
        return "welcome to login module";
    };
    function logout(){
```

```
        return "welcome to logout module";
    };
    function setCredentials(){
        return "welcome to setCredentials module";
    };
    function clearCredentials(){
        return "welcome to clearCredentials module";
    };
    console.log(modules.login(),
                modules.logout(),
                modules.setCredentials(),
                modules.clearCredentials())
  </script>
</html>
```

Ex-7

```
<!DOCTYPE html>
<html>
  <script>
    let obj = {
        mysql  :    ()=>{
            return "mysql connection soon...!"
        },
```

```
        mongodb :   ()=>{
            return "mongodb connection soon...!"
        },
        sqlserver  : ()=>{
            return "sqlserver connection soon...!"
        },
        oracle  : ()=>{
            return "oracle connection soon...!"
        }
    };
    console.log( obj.mysql(),
                 obj.mongodb(),
                 obj.sqlserver(),
                 obj.oracle() );
  </script>
</html>
```

Read the Data From following URL

URL : https://restcountries.eu/rest/v2/all

- above URL technically called as Rest API.
- above URL gives the JSON as Response.
- to know the structure of complex JSON, we have following website

URL : http://jsonviewer.stack.hu/

Structure of Above JSON

- ✓ initially we have JSON Array.
- ✓ JSON Array containes 250 JSON Objects
- ✓ we can iterate JSON Array by using forEach()
- ✓ Each JSON Object containes following keys

  @name

  @capital

  @region

  @population

  @flag

- ✓ Each JSON Object containes following array

  @currencies

- ✓ currencies array 0th position constaines JSON Object with the following keys

  @name

  @code

  @symbol

```
<!DOCTYPE html>
<html>
    <script>
        let countries = "--COPY THE DATA FROM right side URL--";
```

https://restcountries.eu/rest/v2/all (data)

```
document.write(`<table border="1"
```

```
            cellpadding="10px"
            cellspacing="10px"
            align="center">
        <thead style="background-color:gray">
            <tr>
                <th>SNO</th>
                <th>Name</th>
                <th>Capital</th>
                <th>Population</th>
                <th>Code</th>
                <th>Flag</th>
            </tr>
        </thead>
        <tbody>`
);
countries.forEach((element,index)=>{
    document.write(`
        <tr>
            <td>${index+1}</td>
            <td>${element.name}</td>
            <td>${element.capital}</td>
            <td>${element.population}</td>
```

```
                <td>${element.currencies[0].code}</td>
                <td><img width="100px" height="50px"
                    src="${element.flag}"></td>
            </tr>
        `);
    });
  </script>
</html>
```

Assignment::Read the JSON from the following URL and display in the form a table .

url:https://www.w3schools.com/angular/customers.php

```
<!DOCTYPE html>
<html>
  <script>
 let records={ ----above url consume data---};

 document.write(`
        <table border="1"
        cellpadding="2px"
        cellspacing="2px"
        align="center">
```

```
            <thead style="background-color:gray">
            <tr>
            <th>Name</th>
            <th>City</th>
            <th>Country</th>
            </tr>
            </thead>
            </tbody>
    `);
    records.forEach((element,index)=> {
        document.write(`
        <tr>
        <td>${element.records.Name}</td>

        </tr>
        `);
    });
    </script>
</html>
```

AJAX Calls

- if network calls executing independently without effecting any other calls called as AJAX Calls.
- AJAX Calls have the more performance while processing the Requests.
- we will make AJAX Calls by using "jQuery" library.
- we will include jQuery library by using following CDN.

URL : https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js

Ex_1:

- Make the AJAX Call by using following URL.

URL : https://restcountries.eu/rest/v2/all

```
<!DOCTYPE html>
<html>
    <head>
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
    </head>
    <body>
        <script>
            $.ajax({
                method:"GET",
                url:"https://restcountries.eu/rest/v2/all",
                success:(res)=>{
```

```
            console.log(res);
        },
        error:(err)=>{
            console.log(err);
        }
    });
</script>
</body>
</html>
```

Ex_2:

https://www.w3schools.com/angular/customers.php

```
<!DOCTYPE html>
<html>
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js
"></script>
</head>
<body>
<script>
$.ajax({
    method:"GET",
```

```
            url:"https://www.w3schools.com/angular/customers.php",
            success:(posRes)=>{
                //parse is the predefined function in JSON class
                //parse function used to convert the string response to
object response
                console.log(JSON.parse(posRes));
            },
            error:(errRes)=>{
                console.log(errRes);
            }
        });
    </script>
  </body>
</html>
```

Ex_3:

series of AJAX Calls

URL_1: https://restcountries.eu/rest/v2/all

URL_2:https://www.w3schools.com/angular/customers.php

```
<!DOCTYPE html>
<html>
  <head>
```

```
        <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js
"></script>
    </head>
    <body>
        <script>
            $.ajax({
                method:"GET",
                url:"https://restcountries.eu/rest/v2/all",
                success:(posRes1)=>{
                    console.log(posRes1);
                    /*****************************/
                        $.ajax({
                            method:"GET",

url:"https://www.w3schools.com/angular/customers.php",
                            success:(posRes2)=>{
                                console.log(posRes2);
                            },
                            error:(errRes2)=>{
                                console.log(errRes2);
                            }
                        });
```

```
            /*****************************/
        },
        error:(errRes1)=>{
            console.log(errRes1);
        }
    });
</script>
</body>
</html>
```

Ex_4:

Post Request

- we will send the data to the server by using Post Request

URL : http://test-routes.herokuapp.com/test/uppercase

above URL Representing "Post" Request.

above URL carries the data to server in the form of a "JSON Object"

```
<!DOCTYPE html>
<html>
    <head>
        <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js
"></script>
    </head>
```

```
<body>
    <script>
        $.ajax({
            method:"POST",
            url:"http://test-routes.herokuapp.com/test/uppercase",
            data:{message:"angular9"},
            success:(posRes)=>{
                console.log(posRes);
            },
            error:(errRes)=>{
                console.log(errRes);
            }
        });
    </script>
</body>
</html>
```


Promises

- Promises establishes the communication between Producer and Consumer.
- Promises have the two states

  @resolve (success) @reject (failure)

- we will construct promises by using "Promise" class constructor.
- we will consume Promises by using then()

Ex-1

```
<!DOCTYPE html>
<html>
    <script>
        //constructing promise
        let promise1 = new Promise((success,failure)=>{
            success("tomarrow we will discuss async and await keyword");
        });

        //consuming promise
        promise1.then((posRes)=>{
            console.log(posRes);
        },(errRes)=>{
            console.log(errRes);
        });
    </script>
</html>
```

Ex-2

```
<!DOCTYPE html>
<html>
    <script>
      let promise1 = new Promise((resolve,reject)=>{
```

```
        setTimeout(()=>{
            resolve("Hello");
        },5000);
    });

    promise1.then((posRes)=>{
        console.log(posRes);
    },(errRes)=>{
        console.log(errRes);
    });
</script>
</html>
```

Ex-3

```
<!DOCTYPE html>
<html>
    <script>
        let promise1 = new Promise((resolve,reject)=>{
            setTimeout(()=>{
                resolve("Angular9");
            },0);
        });
```

```
let promise2 = new Promise((resolve,reject)=>{
    setTimeout(()=>{
        reject("Fail....!");
    },5000);
});

let promise3 = new Promise((resolve,reject)=>{
    setTimeout(()=>{
        resolve("MongoDB");
    },10000);
});
promise1.then((posRes)=>{
    console.log(posRes);
},(errRes)=>{
    console.log(errRes);
});

promise2.then((posRes)=>{
    console.log(posRes);
},(errRes)=>{
    console.log(errRes);
});
```

```
    promise3.then((posRes)=>{
        console.log(posRes);
    },(errRes)=>{
        console.log(errRes);
    });
    //in above code we are getting result in 0sec,5sec and 10 sec
```

OR

```
    Promise.all([promise1,promise2,promise3])
          .then((posRes)=>{
           console.log(posRes[0]);
           console.log(posRes[1]);
           console.log(posRes[2]);
          },(errRes)=>{
           console.log(errRes);
    });
  </script>
</html>
```

- all() the predefined function in Promise class
- all() used to execute the multiple promises at a time
- all() function gives the consolidated result after max(highest) time in promises.
- in above promises highest time is 10sec
- so, we are able to see the result after 10 sec

- in multiple promises, if anyone promise fails automatically failure result will be displayed.
- never displays the success result.

Ex-4

```
<!DOCTYPE html>
<html>
    <script>
        let promise1 = new Promise((resolve,reject)=>{
            reject("Network call fail");
        });
        promise1.then((posRes)=>{

        },(errRes)=>{
            console.log(errRes);
        });
    </script>
</html>
```

Ex-4

```
<!DOCTYPE html>
<html>
    <script>
        let demo = new Promise((arg1,arg2)=>{
            setTimeout(()=>{
```

```
            arg2("Fail");
        },5000);
    });
    demo.then((res)=>{

    },(err)=>{
        console.log(err);
    });
</script>
</html>
```

Ex-5

```
<!DOCTYPE html>
<html>
    <script>
        let promise1 = new Promise((resolve,reject)=>{
            resolve("Success...!");
            reject("Failure...!");
        });
        promise1.then((posRes)=>{
            console.log(posRes);    //Success...!
        },(errRes)=>{
            console.log(errRes);
```

```
    });
```

- only one state is possible either resolve or reject
- while executing promises order also important
- in above example resolvewil execute
- reason is resolve in first order

```
    </script>
</html>
```

Ex-6

```
<!DOCTYPE html>
<html>
    <script>
        let promise1 = new Promise((resolve,reject)=>{
            setTimeout(()=>{
                resolve("Success");
            },5000);
            setTimeout(()=>{
                reject("Fail");
            },6000);
        });
        promise1.then((posRes)=>{
            console.log(posRes);
        },(errRes)=>{
            console.log(errRes);
```

```
    });
  </script>
</html>
```

async & awiat

- ❖ above keywords introduced in ES9.
- ❖ above keywords used to consume the promises insted of then()
- ❖ if we use async & awiat keywords, application performance will be increased.
- ❖ automatically application readability increses

Ex-1

```
<!DOCTYPE html>
<html>
  <script>
    let promise1 = new Promise((resolve,reject)=>{
      resolve("Hello");
    });

    async function consumePromise(){
      let res = await promise1;
      console.log(res);
    };
   consumePromise();
  </script>
</html>
```

Ex-2

```
<!DOCTYPE html>
<html>
    <script>
        function fun_one(arg1){
            return new Promise((success,failure)=>{
                success(arg1+5);
            });
        };
        function fun_two(arg1){
            return new Promise((resolve,reject)=>{
                resolve(arg1-3);
            });
        };
        async function multiple(){
            let addRes = await fun_one(5);
            let subRes = await fun_two(addRes);
            console.log(addRes,subRes);
        };
        multiple();
    </script>
</html>
```

AJAX with promisses::

URL : https://restcountries.eu/rest/v2/all

Ex-1

```
<!DOCTYPE html>
<html>
  <head>
   <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
  </head>

  <body>
     <script>
        let countries =
                  new Promise((resolve,reject)=>{
            $.ajax({
               method:"GET",
               url:"https://restcountries.eu/rest/v2/all",
               success:(posRes)=>{
                  resolve(posRes);
               },
               error:(errRes)=>{
                  reject(errRes);
```

```
            }
        });
    });
    countries.then((posRes)=>{
        console.log(posRes);
    },(errRes)=>{
        console.log(errRes);
    });
  </script>
 </body>
</html>
```

Ex-2

URL : https://www.w3schools.com/angular/customers.php

```
<!DOCTYPE html>
<html>
  <head>
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js
"></script>
  </head>
  <body>
    <script>
      let customers =
```

```
            new Promise((resolve,reject)=>{
            $.ajax({
                method:"GET",
url:"https://www.w3schools.com/angular/customers.php",
                success:(posRes)=>{
                    resolve(posRes);
                },
                error:(errRes)=>{
                    reject(errRes);
                }
            });
        });
        async function myFun(){
            let res = await customers;
            console.log(res);
        };
        myFun();
    </script>
</body>
</html>
```

Function:

- Particular business logic called as Function.
- Functions are used to reuse the business logic.
- we will define functions by using "function" keyword.
- we have following types of functions
  1) Named Functions
  2) Anonymous Functions/Arrow Functions/CallBack Functions
  3) Rest Parameters in Functions
  4) Optional Parameters in Functions
  5) Default Parameters in Functions

Named Functions

The function with the name called as Named Function.

Syntax:

```
//function definition
function functionname(arguments with datatype):returntype{
    //business logic
};
//call the function
functionname(parameters);
```

Ex1:

```
function fun_one():string{
    return "welcome to named functions";
}
console.log( fun_one );      //function definition [Function: fun_one]
```

```
console.log( fun_one() );    //function output
                             //welcome to named functions
```

Ex2:

```
function fun_one(arg1:string,
                 arg2:string,
                 arg3:string):string{
    return arg1+"<==>"+arg2+"<==>"+arg3;
};
console.log(
    fun_one("Angular9","NodeJS","MongoDB")
);
console.log(
    fun_one("ReactJS","NodeJS","MySQL")
);
console.log(
    fun_one("VueJS","NodeJS","SQLServer")
);
```

Ex-3

```
function fun_one():string{
    return fun_two();
};
function fun_two():string{
```

```
    return "welcome";
};
console.log(fun_one());  //welcome
```

Ex-4

```
function fun_one():any{
    return fun_two;
}
function fun_two():string{
    return "welcome";
};
console.log( fun_one );    //fun_one def
console.log( fun_one() ); //fun_two def
console.log( fun_one()() );  //welcome
```

Ex-5

```
function fun_one(arg1:any,arg2:any,arg3:any):any{
    console.log(arg1,arg2,arg3);
};
function fun_two():string{
    return "Angular9";
};
function fun_three():string{
    return "NodeJS";
```

```
};
function fun_four():string{
    return "MongoDB";
};
fun_one( fun_two(),fun_three(),fun_four() );
//Angular9 NodeJS MongoDB
```

Ex-6

```
function networkCall(param1:any,param2:any):void{
    console.log( param1,param2 );
                    //[Function: success] [Function: error]
    console.log( param1(),param2() );
                    //Got the Rest API Data....! Network Error
};
function success():string{
    return "Got the Rest API Data....!";
};
function error():string{
    return "Network Error";
};
networkCall( success, error );
```

Anonymous Functions

- The function without name called as Anonymous function.

- Anonymous functions also called as arrow functions.
- Arrow functions behaves like CallBack Functions.
- Arrow functions introduced in ES6 Version.
- Arrow functions are more secured.

Syntax:

```
var/let/const variablename =
        (arguments with datatype):returntype=>{
    //business logic
};

variablename(parameters);
```

Ex-1

```
let fun_one = ():void=>{
    console.log("welcome to arrow functions");
};
fun_one();
```

Ex-2

```
let uitech = (sub_one:string,
            sub_two:string,
            sub_three:string):void=>{
    console.log(sub_one,sub_two,sub_three);
};
uitech("Angular","NodeJS","MongoDB");
```

```
uitech("ReactNative","NodeJS","FireBase");
uitech("VueJS","NodeJS","MongoDB");
```

Ex-3

```
let my_fun = ():any=>{
    return ():string=>{
        return "Hello";
    };
};
console.log( my_fun()() );
```

Ex-4

```
let my_fun = ():any=>{
    return ():any=>{
        return ():any=>{
            return "Hello";
        };
    };
};
console.log(
    my_fun()()()
);
```

Ex-5

```
let my_fun = (arg1:any):any=>{
    return ():any=>{
        return (arg2:any):any=>{
            console.log(arg1,arg2);
        };
    };
};
my_fun("Hello_1")()("Hello_2");
```

Ex-6

```
let fun_one = (callback:any):any=>{
    console.log( callback() );
};
fun_one(
    ():any=>{
       return "Hello";
    }
);
```

Ex-7

```
let demo = (arg1:any):any=>{
    setInterval(()=>{
        console.log(arg1());
    },5000);
```

```
};
demo(
    ():any=>{
        return "Welcome";
    }
);
```

Ex-8

```
let my_fun = (arg1:any,callback:any):any=>{
    return callback("Bye Bye to "+arg1);
};


my_fun("Arrow Functions",(res:any):any=>{
    console.log(res);
});
```

Rest parameters

- we will represent rest parameters with the help of "**...**"
- rest parameters released in ES6
- we can store multiple values by using rest parameters(arrays)

Ex-1

```
function myFun(...arg1:any):any{
    console.log(arg1);
};
myFun(10);    //[ 10 ]
```

```
myFun(10,20);  //[ 10, 20 ]
myFun("Angular","Node","ReactJS");
        //[ 'Angular', 'Node', 'ReactJS' ]
```

Ex-2

```
function myFun(...arg1:any):any{
    console.log(arg1);
};
myFun();  //[ ]
```

Ex-3

```
function myFun(...arg1:any,...arg2:any):any{
}
```

- we can't supply more than one rest argument
- rest argument position should be the last in parameters

combination of regular parameter with rest parameter

```
function myFun(arg1:any,...arg2:any):any{
    console.log(arg1,arg2);
};
//myFun();                    //Expected at least 1 arguments, but got 0.
myFun(10);                    //10 []
myFun(10,20);                 //10 [ 20 ]
myFun(10,20,30,40,50);        //10 [ 20, 30, 40, 50 ]
myFun(undefined);             //undefined []
myFun(undefined,undefined);   //undefined [ undefined ]
```

```
myFun(null);                    //null []
myFun(null,null,undefined);     //null [ null, undefined ]
```

optional parameters if functions

- optional parameters we can represent with the help of "**?**"
- we "may / maynot" supply the optional parameters while calling the functions.

Ex-1

```
function myFun(arg1?:any,arg2?:any,arg3?:any):any{
    console.log(arg1,arg2,arg3);
};
myFun();  //undefined undefined undefined
myFun("Angular");    //Angular undefined undefined
myFun("Angular",undefined,"NodeJS");
                    //Angular undefined NodeJS
```

regular parameter with optional parameter

```
function myFun(arg2?:any,arg1:any):any{}
```

- optional parameter should be last in above combination.

```
function myFun(arg1:any,arg2?:any):any{
    console.log(arg1,arg2);
};
myFun();        //Expected 1-2 arguments, but got 0.
```

```
myFun("Hello");   //Hello undefined
myFun(undefined);//undefined undefined
myFun(null);      //null undefined
myFun(null,null);//null null
myFun(undefined,null);   //undefined null
```

Ex-2

```
function myFun(arg1:any,arg2?:any,arg3?:any):any{


}
```

- above application also valid example

optional with rest

```
function myFun(arg1?:any,...arg2:any):any{}
```

regular,rest and optional

```
function fun_one(arg1:any,arg2?:any,...arg3:any):any{
    console.log(arg1,arg2,arg3);
};
fun_one();  //Expected at least 1 arguments, but got 0.
fun_one(undefined,undefined,undefined);
            //undefined undefined [ undefined ]
```

Default Parameters in Functions

- while defining the functions, few arguments are default.

```
function myFun(arg1:string = "Angular",
               arg2:string = "NodeJS",
               arg3:string = "MongoDB"):any{
    console.log(arg1,arg2,arg3);
};
myFun();                                    //Angular NodeJS MongoDB
myFun("ReactJS","AWS","MySQL");             //ReactJS AWS MySQL
myFun("VueJS");                             //VueJS NodeJS MongoDB
myFun(undefined,undefined,"SQLServer");     //Angular NodeJS SQLServer
myFun(null,null,null);                      //null null null
myFun();                                    //Angular NodeJS MongoDB
```


Combinations of functions

- default with regular
- default with optional
- default with rest
- default, regular, optional , and rest

Default parameter with regular parameter

```
function fun_one(arg1:string="Hello_1",arg2:string) :any{
      console.log(arg1,arg2);
};
fun_one();  //Expected 2 arguments, but got 0.
fun_one(undefined,"Hello_2");  //Hello_1 Hello_2
fun_one(null,"Hello_2");       //null Hello_2
fun_one(undefined,undefined);  //Hello_1 undefined
fun_one(null,null);            //null null
```

Default with Optional Parameters

```
 function fun_one(arg1:any="Hello",arg2?:any):any{
     console.log(arg1,arg2);
 };
 fun_one(undefined);    //Hello undefined
 fun_one(undefined,"welcome");   //Hello welcome
fun_one(null);                  //null undefined
 fun_one(null,null);             //null null
```

Default parameter with rest parameter

```
 function fun_one(arg1:string="Hello",...arg2:string[]):void{
    console.log(arg1,arg2);
 };
fun_one();                      //Hello []
```

```
fun_one(undefined);              //Hello []
fun_one(null);                   //null []
fun_one(null,null);              //null [ null ]
fun_one(undefined,undefined);    //Hello [ undefined ]
fun_one(undefined,"Hello");      //Hello [ 'Hello' ]
```

Default,Regular,Optional and Rest.

```
function fun_one(arg1:string="Hello_1",
                 arg2:string,
                 arg3?:string,
                 ...arg4:string[]):void{
    console.log(arg1,arg2,arg3,arg4);
};
//fun_one();   //Expected at least 2 arguments, but got 0.
fun_one(undefined,"Hello_2");  //Hello_1 Hello_2 undefined []
fun_one(undefined,"Hello_2","Hello_3","Hello_4");
                 //Hello_1 Hello_2 Hello_3 [ 'Hello_4' ]
```

OOP

- ❖ TypeScript supports the OOP.
- ❖ TypeScript is the Object Oriented Programming Language.

1) Class

- ❖ collection of variables and functions called as class.
- ❖ collection of objects also called as class.
- ❖ we can create the classes by using "class" keyword.
- ❖ we can create the objects to the classes by using "new" keyword.
- ❖ we can define constructors by using "constructor" keyword.
- ❖ in TypeScript, we have following modifiers.
  - ▪ @public @private @protected
- ❖ The Default Modifier in TypeScript is "public".
- ❖ Recomended modifier for variables is "private".
- ❖ Recomended modifier for functions is "public".

Ex-1

```
class mean{
    private sub_one:string;
    private sub_two:string;
    private sub_three:string;
    constructor(){
        this.sub_one = "Angular9";
        this.sub_two = "NodeJS";
        this.sub_three = "MongoDB";
    };
    public getSubOne():string{
```

```
        return this.sub_one;
    };
    public getSubTwo():string{
        return this.sub_two;
    };
    public getSubThree():string{
        return this.sub_three;
    };
};
let obj:mean = new mean();    //duck typing
console.log(
    obj.getSubOne(),
    obj.getSubTwo(),
    obj.getSubThree()
);
```

Ex-2

parameterized constructor:

```
class class_one{
    private sub:string;
    constructor(private param1:string){
        this.sub = param1;
    };
```

```
    public getSub():string{
        return this.sub;
    };
};
let obj1:class_one = new class_one("MEAN");
console.log(obj1.getSub());

let obj2:class_one = new class_one("MERN");
console.log(obj2.getSub());

let obj3:class_one = new class_one("MEVN");
console.log(obj3.getSub());
```

Ex-3

```
class class_one{
    public myFun():any{
        return new class_two();
    };
};
class class_two{
    public myFun():string{
        return "Hello";
    };
```

```
}; console.log( new class_one().myFun().myFun() );


//create the class_one
class class_one{
```

- declare the variables
- @param1 @param2 @param3
- param1 used to hold the class_two object
- param2 used to catch the class_three object
- param3 used to catch the class_four object

```
    private param1:class_two;

    private param2:class_three;

    private param3:class_four;


    //initilize the above variables with the help of parameterized
constructor
    constructor(private obj1:class_two,
                private obj2:class_three,
                private obj3:class_four){
        this.param1 = obj1;
        this.param2 = obj2;
        this.param3 = obj3;
    };
//create the public function
    public myFun():any{
```

```
        console.log(this.param1.myFun(),
                    this.param2.myFun(),
                    this.param3.myFun());
    };
};

class class_two{
    public myFun():string{
        return "Angular";
    };
};

class class_three{
    public myFun():string{
        return "NodeJS";
    };
};

class class_four{
    public myFun():string{
        return "MongoDB";
    };
```

```
};


let obj:class_one = new class_one( new class_two(),
                                   new class_three(),
                                   new class_four() );
obj.myFun();
```

Inheritance

- Getting the properties from Parent class to Child classes called as Inheritance
- Because of Inheritance code Reusablity happens
- we have types of Inheritance.

@Single Level @Multi Level @Multiple @HyBrid

Single Level

```
class Parent{
    public fun_one():string{
        return "Hello_1";
    };
};
class Child extends Parent{
    public fun_two():string{
        return "Hello_2";
    };
```

```
};
let obj1:Parent = new Parent();
console.log( obj1.fun_one() );  //Hello_1

let obj2:Child = new Child();
console.log( obj2.fun_one(), obj2.fun_two() );  //Hello_1 Hello_2
```

Multi Level Inheritance

```
class class_one{
    public fun_one():string{
        return "Hello_1";
    };
};
class class_two extends class_one{
    public fun_two():string{
        return "Hello_2";
    };
};
class class_three extends class_two{
    public fun_three():string{
        return "Hello_3";
    };
};
```

```
let obj1:class_one = new class_one();
console.log(obj1.fun_one());      //Hello_1

let obj2:class_two = new class_two();
console.log( obj2.fun_one(),obj2.fun_two() );    //Hello_1 Hello_2

let obj3:class_three = new class_three();
console.log( obj3.fun_one(),obj3.fun_two(),obj3.fun_three() );
                                    //Hello_1 Hello_2 Hello_3

let obj4:class_one = new class_two();
console.log(obj4.fun_one());      //Hello_1

let obj5:class_one = new class_three();
console.log(obj5.fun_one());     //Hello_1

let obj6:class_two = new class_three();
console.log( obj6.fun_one(),obj6.fun_two() );     //Hello_1 Hello_2
```

parent class reference can hold the child class objects

child class reference can't hold the parent class objects

multiple inheritance

```
class class_one{}
class class_two{}
class class_three extends class_one,class_two{}
```

multiple inheritance not supported by TypeScript

hybrid inheritance

```
class class_one{}
class class_two extends class_one{}
class class_three extends class_one{}
class class_four extends class_two,class_three{}
```

hybrid inheritance also not supported by TypeScript

polymorphism

- behaves like many called as polymorphism
- polymorphism have following types

    @function overloading

    @function overriding

    Ex-1.

```
function add(arg1:number,arg2:number):any;
function add(arg1:string,arg2:string):any;
function add(arg1:any,arg2:any):any{
    return arg1+arg2;
};
```

```
console.log(add(10,10));
console.log(add("Hello_1","Hello_2"));
```

Ex-2

```
class class_one{
    public display(arg1:string):string;
    public display(arg1:number):number;
    public display(arg1:boolean):boolean;
    public display(arg1:any):any;
    public display(arg1:any):any{
        return arg1;
    };
};
console.log( new class_one().display("Hello") );
console.log( new class_one().display(100) );
console.log( new class_one().display(true) );
console.log( new class_one().display([]) );
```

function overriding

- overriding the parent class functionality with child class functionality called as function overriding
- to implement function overriding inheritance is mandatory.
- where as in case of function overloading inheritance not required.

Ex-1

```
class class_one{
```

```
    public dbData():string{
        return "Oracle Data Soon...!";
    };
};
class class_two extends class_one{
    public dbData():string{
        return "MongoDB Data Soon...!";
    };
};

let obj1:class_two = new class_two();
console.log( obj1.dbData() );     //MongoDB Data Soon...!
let obj2:class_one = new class_one();
console.log( obj2.dbData() );     //Oracle Data Soon...!
```

Encapsulation

- ✓ holding the variables and functions with the help of class called as Encapsulation

Ex.

```
class class_one{
private sub_one:string = "Angular"
        ---
    public getSubOne():string{
        return this.sub_one;
```

```
        };          }
```

Data Hiding

- Hiding the business logic to End Users with the help of particular functions or variables called as Data Hiding.

  Ex.

```
    function add(num1,num2){
        return num1+num2;
    }
add(10,20);
```

Modifiers

- ✓ TypeScript supports the following modifiers
  - @public
  - @private
  - @protected
- ✓ The Default modifier is "public"
- ✓ public members, we can access out of class also.
- ✓ public members, have the scope of project.
- ✓ any where, we can access public members
- ✓ give the less priority to public members, because of security reasons.
- ✓ we can access private members with in the class only
- ✓ while developing project, private modifier plays the keyrole.

- ✓ parent class data(variables and functions) accessable to all child classes with the help of protected modifier.

Ex-1

```
      class class_one{
constructor(public sub_one:string,
                  public sub_two:string,
                  public sub_three:string){


      };
};
let obj1:class_one = new class_one("A","N","M");
console.log( obj1.sub_one,obj1.sub_two,obj1.sub_three );
```

Ex-2

```
class class_one{
   public sub:string;
   constructor(){
      this.sub = "Angular";     };
   public getSub():string{
      return this.sub;
   };
};
class class_two extends class_one{}
let obj:class_two = new class_two();
```

```
console.log( obj.sub, obj.getSub() );
```

private

- ❖ private members accessable with in the class by using "this" keyword.
- ❖ private members we can't access by using "class references".
- ❖ private members won't accessable to "child classes".

Ex-1

```
class class_one{
    private sub:string = "Angular";
};
new class_one().sub;    //Error : Property 'sub' is private and only accessible within class 'class_one'
```

Ex-2

```
class class_one{
    private sub:string = "Angular";
    private getSub():string{
        return this.sub;
    };
};
new class_one().getSub();  //Error : Property 'getSub' is private and only accessible within class 'class_one'.
```

Ex-3

```
class class_one{
```

```
    private sub:string;
    constructor(){
        this.sub = "Angular";
    };
    private fun_one():string{
        return this.sub;
    };
    public fun_two():string{
        return this.fun_one();
    };
};
console.log(
    new class_one().fun_two()
);   //Angular
```

Ex-4

```
class class_one{
    private constructor(){}
};
new class_one();
```

//Constructor of class 'class_one' is private and only accessible within the class declaration.

Ex-5

```
class class_one{
    private fun_one():string{
```

```
        return "Hello";
    };
};
class class_two extends class_one{
};
new class_two().fun_one();    //Error:Property 'fun_one' is private and only accessible within class 'class_one'.
```

Ex-6

private class class_one{};

NOTE:- class level allowed keywords are "export" "default"

in class level we can't denote any other keywords like public, private and protected,.....

Ex-7

```
class class_one{
    private sub:string;
    constructor(){
        this.sub = "Angular";
    };
    private getSub():string{
        return this.sub;
    };
    public getData():string{
        return this.getSub();
```

```
    };
};
class class_two extends class_one{};
console.log( new class_two().getData() );  //Angular
```

Ex-8

```
class class_one{
    private constructor(){}
    static obj:any = new class_one();
    public fun_two():string{
        return "Hello...!";
    };
};
console.log(  class_one.obj.fun_two() );   //Hello...!
```

protected

- protected members we can't refer with the help of objects
- protected members we can refer in child classes with the help of "this" keyword.

Ex-1

```
class class_one{
    protected data:number = 100;
    protected constructor(){
        this.data = 200;
    };
```

```
    protected myFun():any{
        return this.data;
    };
};
```

protected allowed at variables,constructor,and function level

Ex-2

```
class class_one{
    protected sub:string = "Hello"
};
new class_one().sub;    //Error:Property 'sub' is protected and only accessible within class 'class_one' and its subclasses.
```

Ex-3

```
class class_one{
    protected sub:string;
    constructor(){
        this.sub = "Angular";
    };
};
class class_two extends class_one{}
let obj:class_two = new class_two();
obj.sub;    //Error : Property 'sub' is protected and only accessible within class 'class_one' and its subclasses.
```

Ex-4

```
class class_one{
    protected fun_one():string{
 return "Hello";
    };
    public fun_two():string{
        return this.fun_one();
    };
};
console.log( new class_one().fun_two() );    //Hello
```

Ex-5

```
class class_one{
    protected constructor(){}
};
new class_one();  //Error:Constructor of class 'class_one' is protected and only accessible within the class declaration.
```

Ex-6

```
class class_one{
    constructor(protected arg1:any){
        console.log(arg1);
    }
};
new class_one("Hello");  //Hello
```

static

- static is the keyword
- we can access static members directly by class names
- memory will allotted in heap area for static members
- we can't access static members by using class objects
- we can't initilize static members by using constructors

Ex-1

```
class class_one{
    static data:any;
    //static constructor(){}
    static myFun():any{}
}
```

static applicable to variable and function level

Ex-1

```
class class_one{
    static data:any;
    constructor(){
        this.data = "Hello";
    }
```

```
};
```

we can't initilize static members by using constructors

Ex-1

```
class class_one{
    static data:any = "Hello"
};
new class_one().data;
//we can't access static members by using class objects
```

Ex-2

```
class class_one{
    static arg1:any = "Hello";
    static myFun():any{
        return this.arg1;
    };
};
console.log( class_one.arg1, class_one.myFun() );
```

readonly

- readonly is the keyword
- we can only read the data, but we can't update the data
- we can initlize with the help of constructors

Ex-1

```
class class_one{
```

```
    readonly data:any="Hello";
};
console.log( new class_one().data );  //Hello
new class_one().data = "Welcome";  //Error:Cannot assign to 'data' because it is a read-only property.
```

Ex-2

```
class class_one{
    readonly data:any;
    constructor(){
        this.data = "Hello";
    }
};
console.log( new class_one().data );  //Hello
```

interfaces

- when ever we don't know the implementations then we will choose interfaces.
- implementations known by either JSON or classes.
- we will define interfaces by using "interface" keyword.
- we will implement interfaces by using "implements" keyword in classes
- interfaces are optional in TypeScript

Ex-1

```
interface interface1{
```

```
    data:string;
};
let obj:interface1 = {
    data:"Hello"
};
console.log( obj.data );
```

Ex-2

```
interface interface1{
    myFun():string;
};
let obj:interface1 = {
    myFun : ():string=>{
        return "Hello";
    }
};
console.log( obj.myFun() );  //Hello
```

Ex-3

```
interface interface1{
    sub_one:string;
    getSubOne():string;
};
let obj:interface1 = {
```

```
    sub_one   : "Angular",
    getSubOne : ():string=>{
        return "Welcome to "+obj.sub_one;
    }
};
console.log( obj.sub_one, obj.getSubOne() );
```

Ex-4

single level inheritance in interfaces

```
interface interface1{
    mean:string;
};
interface interface2 extends interface1{
    mern:string;
};
class class_one implements interface2{
    mern:string;
    constructor(arg1:string,arg2:string){
        this.mean = arg1;
        this.mern = arg2;
    }
};
let obj1:class_one = new class_one("AngularJS","ReactJS");
```

```
console.log(obj1.mean,obj1.mern);


let obj2:class_one = new class_one("Angular","React");
console.log(obj2.mean,obj2.mern);
```

Ex-5

multi level inheritance

```
interface interface1{
    fun_one():string;
};
interface interface2 extends interface1{
    fun_two():string;
};
interface interface3 extends interface2{
    fun_three():string;
};
class my_class implements interface3{
    fun_one():string{
        return "fun_one";
    }
    fun_two():string{
        return "fun_two";
    }
```

```
    fun_three():string{
        return "fun_three";
    }
};
let obj:my_class = new my_class();
console.log(obj.fun_one(),
            obj.fun_two(),
            obj.fun_three());
```

Ex-6

multiple inheritance

```
interface interface1{
    sub_one:string;
};
interface interface2{
    sub_two:string;
}
interface interface3 extends interface1,interface2{
    sub_three:string;
};
class class_one implements interface3{
    sub_one:string="Hello_1";
    sub_two:string="Hello_2";
```

```
    sub_three:string="Hello_3";
    constructor(){
        this.sub_one = "Welcome_1";
        this.sub_two = "Welcome_2";
        this.sub_three = "Welcome_3";
    }
};
let obj:class_one = new class_one();
console.log( obj.sub_one,obj.sub_two,obj.sub_three );
//Welcome_1 Welcome_2 Welcome_3
```

Abstract Classes

- ❖ when ever we know the partial implementations, then we will choose Abstract Classes.
- ❖ we will define Abstract classes by using "abstract" keyword.

Ex-1

```
abstract class class_one{
    public fun_one():string{
        return "fun one !!!";
    };
    public abstract fun_two():string;
};
class class_two extends class_one{
    public fun_two():string{
```

```
        return "fun two !!!";
    };
};
let obj:class_two = new class_two();
console.log( obj.fun_one(),obj.fun_two() );
//fun one !!! fun two !!!
```

Ex-2

```
abstract class class_one{
    abstract fun_one():string;
};
abstract class class_two extends class_one{
    abstract fun_two():string;
};
abstract class class_three extends class_two{
    abstract fun_three():string;
};
class my_class extends class_three{
    fun_one():string{
        return "fun one !!!";
    };
    fun_two():string{
        return "fun two !!!";
```

```
    };
    fun_three():string{
        return "fun three !!!";
    };
};
let obj:my_class = new my_class();
console.log( obj.fun_one(), obj.fun_two(), obj.fun_three() );
//fun one !!! fun two !!! fun three !!!
```

Ex-3

```
interface interface1{
    fun_one():string;
};
abstract class class_one implements interface1{
    fun_one():string{
        return "fun one !!!";
    };
    abstract fun_two():string;
};
class class_two extends class_one{
    fun_two():string{
        return "fun two !!!";
    }
```

```
};
let obj:class_two = new class_two();
console.log( obj.fun_one(), obj.fun_two() );
//fun one !!! fun two !!!
```

Enum

- Enum used to create the custom datatype
- "enum" is the predefined keyword
- we have following enum types
- @number @string @hetrogeneous

Ex-1

```
enum Holidays{
    KEY1,
    KEY2,
    KEY3,
    KEY4
}
console.log(Holidays);
console.log(Holidays.KEY1,
            Holidays.KEY2,
            Holidays.KEY3,
            Holidays.KEY4);
//0 1 2 3
```

Ex-2

```
enum MyDataType{
    KEY1 = 10,
    KEY2,
    KEY3,
    KEY4
};
console.log( MyDataType.KEY1,
             MyDataType.KEY2,
             MyDataType.KEY3,
             MyDataType.KEY4 );   //10 11 12 13
```

Ex-3

```
enum Holidays{
    C = 0,
    P = 2,
    D = 1,
    DI = 0.25
};
console.log(Holidays.C,
            Holidays.P,
            Holidays.D,
            Holidays.DI);  //0 2 1 0.25
```

Ex-4

```
enum MyDataType{
    HELLO = 1
};
function my_fun(arg1:any):void{
    console.log( arg1.HELLO );
};
my_fun(MyDataType);    //1
```

Ex-5

```
enum MyDataType{
    KEY1 = fun_one()
};
function fun_one(){
    return 100;
};
console.log( MyDataType.KEY1 );  //100
```

Ex-6

```
enum MyDataType{
    KEY1 = my_fun("Hell"),
    KEY2 = KEY1*10
}
function my_fun(arg1:string):any{
```

```
    if(arg1 === "Hello"){
        return 1;
    }else{
        return 2;
    }
};
console.log( MyDataType.KEY1, MyDataType.KEY2 );
//2 20
```

Ex-7

```
enum MyEnum{
    str_1 = "hello_1",
    str_2 = "hello_2",
    str_3 = "hello_3",
    str_4 = "hello_4"
};
console.log( MyEnum );
```

Ex-8

```
enum MyEnum{
    str_1="Hello",
    str_2="Welcome"
};
console.log( MyEnum.str_1,MyEnum.str_2 );//Hello Welcome
```

Ex-9

```
enum MyEnum{
    str_1 = "Hello",
    str_2 = my_fun()  //Computed values are not permitted
in an enum with string valued members.
}
function my_fun():any{
    return "Hello";
};
```

functional assignment not possible in string enum

functional assignment possible in number enum

hetrogeneous enum

- collection of "number" and "string" enums called as hetrogeneous enum

Ex-10

```
enum htrEnum{
    NUMBER = 1,
    STR_1 = "Hello_1",
    CON   = 2,
    STR_2 = "Hello_2"
};
enum numEnum{
    NUMBER = 1
```

```
}
enum strEnum{
    STR = "HELLO"
};
console.log( typeof(htrEnum),
            typeof(numEnum),
            typeof(strEnum) );
//object object object
```

Module

- Logical Grouping of Similar Functionalities Called as Module
- Module increases the Application Readability
- we can create Modules by using "module" keyword.
- Modules have the global scope

Ex.1

demo.ts

```
module Transactions{
    export function credit():string{
        return "Credit !!!";
    };
    export function debit():string{
        return "Debit !!!";
    };
```

```
    export function ministatement():string{
        return "MiniStatement !!!";
    };
};
```

app.ts

```
//import Transactions Module
///<reference path="./demo.ts" />
console.log( Transactions.credit(),
             Transactions.debit(),
             Transactions.ministatement() );

//Execution
>tsc --target es6 app.ts --outfile res.js
>node res.js
```

Ex.2

demo.ts

```
module Staff{
    export const EMPLOYESS:number=1000;
    export const MANAGERS:number = 500;
    export const CLERKS:number = 400;
};
```

app.ts

```
///<reference path="./demo.ts" />
console.log(
    Staff.EMPLOYESS,
    Staff.MANAGERS,
    Staff.CLERKS
);
```

Ex.3

demo.ts

```
module Demo{
    export let obj:any = {host:"localhost",
                          user:"root",
                          password:"root",
                          database:"workshop"};
};

///<reference path="./demo.ts" />
console.log( Demo.obj.host,
             Demo.obj.user,
             Demo.obj.password,
             Demo.obj.database );
```

Ex.4

demo.ts

```
function fun_one():string{
    return "Hello";
};
module Demo{
    export function fun_two():string{
        return fun_one();
    };
};
```

app.ts

```
///<reference path="./demo.ts" />
console.log( Demo.fun_two() );
```

Between two file exchange the data

Ex.5

demo.ts

```
module Demo{
    export function fun_one(arg1:string):string{
        return "welcome to "+arg1;
    }
};
```

app.ts

```
///<reference path="./demo.ts" />
console.log(Demo.fun_one("Angular"));
```

One module exporting another module

Ex.6

demo.ts

```
module Demo1{
    export module Demo2{
        export let wish:string = "Hello";
    }
};
```

app.ts

```
///<reference path="./demo.ts" />
console.log( Demo1.Demo2.wish );
```

Ex.7

test1.ts

- in this file we will create "Module2"
- "Module2" receives the data from "Module1"
- "Module2" receives "string" data.
- "Module2" reverse the string and returns to "Module1"

```
module Module2{
    export function reverseStr(arg1:string):string{
```

```
        //arg1 is string
        //string-->array
        //array -->reverse
        //array -->string
        return Array.from(arg1).reverse().join("");
    };
};
```

test2.ts

I. in this file we will create Module1

II. "Module1" receives data from "app.ts" file and sends to "Module2"

III. import Module2

```
///<reference path="./test1.ts" />
module Module1{
    export function my_fun(arg1:string):string{
        return Module2.reverseStr(arg1);
    };
};
```

app.ts

```
///<reference path="./test2.ts" />
console.log( Module1.my_fun("Hello") );
```

Ex.8

demo.ts

```
module Demo{
    export class class_one{
        arr:any[] = [10,20,30,40,50];
    }
};
```

app.ts

```
///<reference path="./demo.ts" />
  console.log(new Demo.class_one().arr);
```

.....

To find the samation of array use

```
      reduce((fv,sv)=>{
              return fv+sv;
      })
```

.....

```
console.log(
    new Demo.class_one().arr.reduce((fv,sv)=>{
        return fv+sv;
    })
);
```

.....

Tuple

- Collection of Hetrogeneous Elements called as Tuple.

Ex.1

```
var emp = [111,"e_one",10000];
console.log(emp);                    //[ 111, 'e_one', 10000 ]
console.log( emp[0],emp[1],emp[2] );  //111 e_one 10000
emp.push(222);
console.log(emp);                    //[ 111, 'e_one', 10000, 222 ]
emp.pop();
console.log(emp);                    //[ 111, 'e_one', 10000 ]
emp[2] = 100000;
console.log(emp);                    //[ 111, 'e_one', 100000 ]
```

Destructing the Tuple

```
let[e_id,e_name,e_sal]=emp;
console.log(e_id,e_name,e_sal);      //111 e_one 100000
for(let value of emp){
    console.log(value);              //111
                                     //e_one
                                     //100000
};
```

Ex.2

```
let tuple1:any = [111];
let tuple2:any = ["e_one"];
let tuple3:any = [10000];
console.log( tuple1.concat(tuple2,tuple3) );
//[ 111, 'e_one', 10000 ]
```

Ex.3

```
let tuple = [10,20,30,40,50,60,70,80,90,100];
tuple.splice(4,2);
console.log(tuple);    //[10,20,30,40,70,80,90,100]
tuple.splice(4,2);
console.log(tuple);    //[ 10, 20, 30, 40, 90, 100 ]
tuple.splice(4,0,50,60,70,80);
console.log(tuple);   //[10,20,30,40,50,60,70,80,90,100]
```

Ex.4

```
let tuple = [10,20,30,40,50,60];
console.log( tuple.slice(2,4) );    //[ 30, 40 ]
console.log( tuple.slice(4,5) );    //[ 50 ]
console.log( tuple.slice(2) );      //[ 30, 40, 50, 60 ]
console.log( tuple.slice(2,-1) );   //[ 30, 40, 50 ]
```

union

- assigning the mutiple datatypes to variables called as union

Ex.1

```
let data:number|string|boolean;
data = 100;
console.log(data);    //100
data = "Hello";
console.log(data);    //Hello
data = true;
console.log(data);    //true
```

Ex.2

```
function myFun(arg1:number|string):void{
    console.log(arg1);
};
myFun(100);          //100
myFun("Hello");          //Hello
```

Generics

- Generics mainly for Type Safety in TypeScript.
- By Using Generics we can increse the code readability of application.

Ex.1

```
function myFun<A>(arg1:A):void{
    console.log(arg1);
};
myFun<number>(100);                    //100
myFun<string>("Hello");                //Hello
myFun<boolean>(true);                  //true
myFun<string>(100);    //Error:Argument of type '100' is not assignable to parameter of type 'string'.
```

Ex.2

```
function myFun<A,B>(arg1:A,arg2:B):void{
    console.log(arg1,arg2);
};
myFun<number,number>(10,10);
myFun<string,string>("Hello_1","Hello_2");
myFun<boolean,boolean>(true,false);
myFun<number,string>(100,"Hello");
myFun<number[],string[]>([10],["Hello"]);
let enum1={
```

```
    X:1
};
let enum2={
    X:2
};
myFun<any,any>(enum1,enum2);
```

Ex.3

```
function myFun<A>(arg1:A,arg2:number):void{
    console.log(arg1,arg2);
};
myFun<number>(10,10);
myFun<string>("Hello",10);
myFun<any[]>([10],10);
```

Ex.4

```
class class_one<A,B>{
    private arg1:A;
    private arg2:B;
    constructor(param1:A,param2:B){
        this.arg1 = param1;
        this.arg2 = param2;
    };
    public display():void{
```

```
        console.log(this.arg1,
                    this.arg2);
    };
};
let obj = new class_one<number,number>(10,10);
obj.display();

let obj1 = new class_one<string,string>("Hello","Hello");
obj1.display();
```

never

- never is the special datatype in TypeScript
- if we don't know exact result of functions, then we will choose never datatype

Ex.1

```
function myFun():never{
    while(true){
        console.log("Hello");
    }
};
myFun();
```

```
==============================================
JavaScript:::
                Arrays Manupulation
/*
    map()
        - it is used to manipulate the each and every element in array


console.log(
    [10,20,30,40,50].map((element,index)=>{
        return element+"$";
    })
);  //[ '10$', '20$', '30$', '40$', '50$' ]
*/


/*
    filter()
        - it is used to create the new array based on condition


console.log(
    [10,20,30,40,50].filter((element,index)=>{
        return element>=30;
    })
```

```
);  //[ 30, 40, 50 ]
*/

/*
    findIndex()
        - this function used to know the index of particular element in
array
*/
/*
let arr = [10,20,30,40,50];
let i = arr.findIndex(
                    (element,index)=>{
    return element == 30;
});
arr.splice(i,1);
console.log(arr);  //[ 10, 20, 40, 50 ]



let arr = [{e_id:111},
           {e_id:222},
           {e_id:333},
           {e_id:444},
```

```
        {e_id:555}];
let i = arr.findIndex((element,index)=>{
    return element.e_id == 333;
});
arr.splice(i,1);
console.log(arr);    //[ { e_id: 111 }, { e_id: 222 }, { e_id: 444 }, { e_id: 555 }
]
*/



/*
let arr = [10,20,30,10,40,20,30];
arr.forEach((element,index)=>{
    console.log( arr.indexOf(element), index );
});
//0 1 2 0 4 1 2
//0 1 2 3 4 5 6
*/


/*
let arr = [1,2,3,1,2,3,1,2];
console.log(
    arr.filter((element,index)=>{
```

```
        return arr.indexOf(element) == index;
    })
);  //[1,2,3]
*/
/*
//Deep Cloning
//both the arrays are pointing to same memory location
let arr1 = [1,2,3];
console.log(arr1);    //[ 1, 2, 3 ]
let arr2 = arr1;
console.log(arr1);    //[ 1, 2, 3 ]
console.log(arr2);    //[ 1, 2, 3 ]

arr1.push(4);
console.log(arr1);    //[ 1, 2, 3, 4 ]
console.log(arr2);    //[ 1, 2, 3, 4 ]
*/
/*
//shallow cloning
//both the arrays occupies the different memory locations
//we will implement shallow cloning by using spread operator
let arr1 = [10,20,30];
```

```
console.log(arr1);    //[ 10, 20, 30 ]
let arr2 = [...arr1];
arr1.push(40);
console.log(arr1);    //[ 10, 20, 30, 40 ]
console.log(arr2);    //[ 10, 20, 30 ]
*/

/*
console.log(
    [10,20,30].reverse()
);  //[30, 20, 10]

console.log(
    ["Angular","Node"].reverse()
);    //[ 'Node', 'Angular' ]
*/

console.log(
    [10,50,20,40,30].sort((num1,num2)=>{
        return num1-num2;
    })[1]
);    //20 (second minimum element)
```

```
console.log(
    [10,50,20,40,30].sort((num1,num2)=>{
        return num2-num1
    })[1]
);  //40 (second maximum element)
```