1.Install TypeScript

a. Install Node js
b. Installed typescript from npm in vs code.
c. Opened git bash terminal to open the type script file.
d. A js file be made for the typescript file.

2. Basics of TypeScript

a. Array- An array is a collection of similar data elements stored at contiguous memory locations.

Input:

```
var arr:Array<number>=[1,2,3,4,5,6];
console.log(arr);
//if we want an array to have both number and string as its elements then;
var arr1:(string|number)[]=[ "Goutam","Alwar", "Surya" ];
arr1.push(10);
console.log(arr1);
```

Output

```
▼(6) [1, 2, 3, 4, 5, 6]
   0: 1
   1: 2
   2: 3
   3: 4
   4: 5
   5: 6
   length: 6
  ▶[[Prototype]]: Array(0)
▼(4) ['Goutam', 'Alwar', 'Surya', 10]
   0: "Goutam"
   1: "Alwar"
   2: "Surya"
   3: 10
   length: 4
  ▶[[Prototype]]: Array(0)
```

b. Tuples

If we need to have an array in which the datatypes of the inputs is to be fixed to some datatype we can use tuples.

Input

```
var tupple: [name: string, id: number];
tupple = ["Goutam", 720];
console.log(tupple);
//but it can provide no protection beyond index 1
tupple.push("Alwar");
console.log(tupple);
```

Output

```
▼ (4) ['Goutam', 'Alwar', 'Surya', 10]
    0: "Goutam"
    1: "Alwar"
    2: "Surya"
    3: 10
    length: 4
  ▶ [[Prototype]]: Array(0)
```

c. Type aliasing

Type aliasing allows defining types with custom names.

Input

```
console.log("hello world");
type User={
    name: string;
    email:string;
    phoneNumber:number;
    creditCard ?: string;// ? makes the entry of detils for the credit card optional i.e an error won't popup when card details are not sent.
}

var  createUser =function(user:User):User{
    var obj={name : user.name,
    email:user.email,
    phoneNumber:user.phoneNumber,}
    return obj;
}
var userList=[
    createUser({name:"RITESH", email: "kishan", phoneNumber:1112222}),
    createUser({name:"Surya", email: "Bhoi", phoneNumber:1112222}),
    createUser({name:"Goutam", email: "Alwar", phoneNumber:1112222, })
]
console.log(userList);
```

Output

```
▼(3) [{…}, {…}, {…}]
  ▶0: {name: 'RITESH', email: 'kishan', phoneNumber: 1112222}
  ▶1: {name: 'Surya', email: 'Bhoi', phoneNumber: 1112222}
  ▶2: {name: 'Goutam', email: 'Alwar', phoneNumber: 1112222}
   length: 3
  ▶[[Prototype]]: Array(0)
```

3. Classes

Class keyword is used in typescript is used to make a piece of reusable code of which multiple objects can be made. A class can implement multiple interfaces but can extend to only one other class.

Input

```
class multiply {
    // num1:number;
    // num2:number;
    // constructor(num1:number,num2:number) {
    //     this.num1=num1;
    //     this.num2=num2;
    // }
    constructor(
        public num1:number,
        public num2:number
    ){}
    result(num1:number,num2:number):number{
        return num1*num2;
    }
}
var new_object= new multiply(5,20);
console.log(new_object);//returns the multiply object
console.log(new_object.result(5,10));//invoking the result functon of the muliply class.
```

Output

```
▼ multiply {num1: 5, num2: 20}
    num1: 5
    num2: 20
  ▶ [[Prototype]]: Object
50
>
```

4. Interfaces

Interfaces give a syntax for the implementing classes i.e the implementing classes of the interface must define its members.

Input

```
//interfaces defines the syntax for the clasess which implements it i.e the class that implements the
//  interface must provide implementation for all the methods and variables mentioned in the interface
// Interfaces cannot be instantiated on their own
//we can also add futher methods or vriables to a interface.
interface Employee
{
    name:string,
    employee_id:number
}
interface Employee
{
    work(position:string):string;
}

var company:Employee={
    name:"Ritesh", employee_id:277, work:(position:"developer")=>{
        return "software-developer"
    }

}
console.log(company);
```

Output

```
▼ {name: 'Ritesh', employee_id: 277, work: ƒ}
    employee_id: 277
    name: "Ritesh"
  ▶ work: ƒ (position)
  ▶ [[Prototype]]: Object
```

5. Enum- It is used to define the set of **named constants**, i.e., a collection of related values. TypeScript supports both **numeric** and **string-based** enums. We can define the enums by using the **enum** keyword.

Input

```
//enum is used to group a related variables i.e. to define a set of named constants which cannot be chang
// enum Status
// {
//     Active,
//     Inactive,
//     Onhold
// }
//by default active is assigned with numeric value of 0 and rest have an increment of 1
//console.log(Status.Inactive);//output is 1
// It can be changed to have uniques numeric values
enum Status
{
    Active= 15,
    Inactive= 45,
    Onhold=22
}
console.log(Status.Inactive)//output is 45

// the variables can also be assigned to have string values
enum CardinalDirections {
    North = 'North',
    East = "East",
    South = "South",
    West = "West"
  };
  // logs "North"
  console.log(CardinalDirections.North);
```

Output

```
45
North
```

6. Union

we can define a variable which can have multiple types of values. In other words, TypeScript can combine one or two different types of data (i.e., number, string, etc.) in a single type, which is called a union type. Union types are a powerful way to express a variable with multiple types.

Input

```
//union type
var id:number;// now id can only have data of type number.
id=10;
id="Alwar";//error
//now if we want a variable to store both number or string then,
var password:string|number;
password=10;
password="Alwar"//no error
```

7. Generics

While using any is certainly generic in that it will cause the function to accept any and all types for the type of arg, we actually are losing the information about what that type was when the function returns. If we passed in a number, the only information we have is that any type could be returned.

Input

```
//Generics
function element<type>(arr:Array<type>):type[]
{
    return arr;
}
var arr2:Array<number>=[1,2,3,4,5,6,7,8,9]
console.log(element(arr2));

var arr3:Array<string>=["hello","world","goutam","alwar"]
console.log(element(arr3));
```

Output

```
▼(9) [1, 2, 3, 4, 5, 6, 7, 8, 9]
  0: 1
  1: 2
  2: 3
  3: 4
  4: 5
  5: 6
  6: 7
  7: 8
  8: 9
  length: 9
  ▶[[Prototype]]: Array(0)
▼(4) ['hello', 'world', 'goutam', 'alwar']
  0: "hello"
  1: "world"
  2: "goutam"
  3: "alwar"
  length: 4
  ▶[[Prototype]]: Array(0)
```