What is typescript?

1. TypeScript is an open-source language which builds on JavaScript, one of the world's most used tools, by adding static type definitions.TypeScript is a language that is a superset of JavaScript. JS syntax is therefore legal TS. The main benefit of TypeScript is that it can highlight unexpected behavior in your code, lowering the chance of bugs.
2. TypeScript code is written in a file with .ts extension and then compiled into JavaScript using the compiler. You can write the file in any code editor and the compiler needs to be installed on your platform, 3.the command tsc .ts compiles the TypeScript code into a plain JavaScript file.

What are Types(Data types) in TypeScript?

1. Boolean -The most basic datatype is the simple true/false value, which JavaScript and TypeScript call a boolean value. -let isDone: boolean = false;Try
2. Number -As in JavaScript, all numbers in TypeScript are either floating point values or BigIntegers. These floating point numbers get the type number, while BigIntegers get the type bigint. In addition to hexadecimal and decimal literals, TypeScript also supports binary and octal literals introduced in ECMAScript 2015.

```
- let decimal: number = 6;
- let hex: number = 0xf00d;
- let binary: number = 0b1010;
- let octal: number = 0o744;
- let big: bigint = 100n;Try
```

3. String

- Another fundamental part of creating programs in JavaScript for webpages and servers alike is working with textual data. As in other languages, we use the type string to refer to these textual datatypes. Just like JavaScript, TypeScript also uses double quotes (") or single quotes (') to surround string data.

```
let color: string = "blue";
color = "red";Try
You can also use template strings, which can span multiple lines and have
embedded expressions. These strings are surrounded by the
backtick/backquote (`) character, and embedded expressions are of the form
${ expr }.
```

4 .Array

- TypeScript, like JavaScript, allows you to work with arrays of values. Array types can be written in one of two ways. 1.use the type of the elements followed by [] to denote an array of that element type:

```
let list: number[] = [1, 2, 3];Try
The second way uses a generic array type, Array<elemType>:

let list: Array<number> = [1, 2, 3];
```

5 .Tuple

- Tuple types allow you to express an array with a fixed number of elements whose types are known, but need not be the same. For example, you may want to represent a value as a pair of a string and a number:

```
// Declare a tuple type
let x: [string, number];
// Initialize it
x = ["hello", 10]; // OK
When accessing an element with a known index, the correct type is
retrieved:
```

6. Enum

- addition to the standard set of datatypes from JavaScript is the enum. As in languages like C#, an enum is a way of giving more friendly names to sets of numeric values.

```
enum Color {
  Red,
  Green,
  Blue,
}
let c: Color = Color.Green;Try
By default, enums begin numbering their members starting at 0.
```

7. Unknown

- We may need to describe the type of variables that we do not know when we are writing an application. These values may come from dynamic content – e.g. from the user – or we may want to intentionally accept all values in our API. In these cases, we want to provide a type that tells the compiler and future readers that this variable could be anything, so we give it the unknown type.

```
let notSure: unknown = 4;
notSure = "maybe a string instead";
notSure = false;
```

8. Any

- In some situations, not all type information is available or its declaration would take an inappropriate amount of effort. These may occur for values from code that has been written without TypeScript or a 3rd party library. In these cases, we might want to opt-out of type checking. To do so, we label these values with the any type:

```
declare function getValue(key: string): any;
// OK, return value of 'getValue' is not checked
const str: string = getValue("myString");

The any type is a powerful way to work with existing JavaScript, allowing
you to gradually opt-in and opt-out of type checking during compilation.
```

9.Void

- void is a little like the opposite of any: the absence of having any type at all. You may commonly see this as the return type of functions that do not return a value:

```
function warnUser(): void {
  console.log("This is my warning message");
}
```

10.Never

- The never type represents the type of values that never occur. For instance, never is the return type for a function expression or an arrow function expression that always throws an exception or one that never returns.
- The never type is a subtype of, and assignable to, every type;
- however, no type is a subtype of, or assignable to, never (except never itself). Even any isn't assignable to never.

Some examples of functions returning never:

```
// Function returning never must not have a reachable end point
function error(message: string): never {
  throw new Error(message);
}

// Inferred return type is never
function fail() {
  return error("Something failed");
}
```

Type assertions

Type assertions are a way to tell the compiler “trust me, I know what I’m doing.” A type assertion is like a type cast in other languages, but it performs no special checking or restructuring of data. Type assertions have two forms. One is the as-syntax:

```
let someValue: unknown = "this is a string";

let strLength: number = (someValue as string).length;
The other version is the “angle-bracket” syntax:

let someValue: unknown = "this is a string";

let strLength: number = (<string>someValue).length;Try
The two samples are equivalent. Using one over the other is mostly a choice
of preference
```

What are the Differences between TypeScript and JavaScript?

- TypeScript stands in an unusual relationship to JavaScript.

1. TypeScript offers all of JavaScript’s features, and an additional layer on top of these: TypeScript’s type system.
2. For example, JavaScript provides language primitives like string, number, and object, but it doesn’t check that you’ve consistently assigned these. TypeScript does.
3. This means that your existing working JavaScript code is also TypeScript code.
4. The main benefit of TypeScript is that it can highlight unexpected behavior in your code, lowering the chance of bugs, which javascript cant do

- difference between ts and js:

1. TS-TypeScript is an Object-Oriented language

JS-JavaScript is a Scripting language

2. TS-It has a feature known as Static typing

JS-It does not have static typing

3. TS-TypeScript gives support for modules

JS-JavaScript does not support modules

4. TS-It supports optional parameter function

- JS-It does not support optional parameter function

How do you compile TypeScript files?

```
    tsc <filename>.ts; and to run the output file , node <filename>.js
```

How to install TypeScript?

```
sudo npm install -g typescript
```

Who developed Typescript and what is the current stable version available?

- TypeScript
- Developer Microsoft
- First appeared 1 October 2012
- Stable release 4.0.2 / 20 August 2020

How to write a function in typescript?

- TypeScript functions can be created both as a named function or as an anonymous function.
- 1 . function add(x:number,y:number) :number{ return x + y;} 2 . let add = function(x:number,y:number) :number{return x + y;}

additional features in typescript in comparison to javascript?

- new features in TS are:
- A. .data types
- B. compilation of code
- C. introduced class as an OOP concept
- D. it is statically typed language
- E. All of the above
- ans : E,
    - as TypeScript is a language that is a superset of JavaScript,TypeScript offers all of JavaScript's features, and an additional layer on top of these: TypeScript's type system.

What are the Benefits of using TypeScript?

- Typescript is a strict, typed superset of Javascript developed by Microsoft.
- the major benefits of using Typescript in applications.

1. Static Typing The main benefit of Typescript is that it offers the ability to add static types to your Javascript code. Javascript is a dynamically typed language, Static typing allows you to catch errors earlier in the debugging process.

2. Typescript Error Catching the error instantly allows me to save time and check why my input is not the type I think it is, rather than having to trace my steps back much later in the process without being sure where the error is occurring.

3. static typing Optional

- it's worth emphasizing that the static typing available in Typescript is optional, which is a nice benefit. with Typescript you only have to use the static typing where you think it will be beneficial.

4. New Features + Browser Compatibility

- Typescript mixes the old with the new in the best way possibility.
- You can use newer features from ES6, ES7, and beyond, and the compiler will convert them to ES5 (or whatever you specify). This allows you to make use of newer features that will be the future of Javascript without having to worry about compatibility. The finished code produced by the compiler will be backwards compatible with ES5 and will work just fine with older browsers.

5. Enhanced IDE Experience

- Due to the static types, several IDEs are able to offer even more predictive assistance than usual (with an instance of a class and its attributes, for example). -e.g. Atom is one of the IDEs that has a great Typescript package worth checking out.

What are the different ways of declaring a Variable?

- The type syntax for declaring a variable in TypeScript is to include a colon (😃 after the variable name, followed by its type. Just as in JavaScript, we use the var keyword to declare a variable.
- fundamentally there are two ways,
- implicit and
- explicit
- based on which , while declaring a variable, you have four options –

1. Declare its type and value in one statement.

```
 var name:string = ”mary”

//The variable stores a value of type string
```

2. Declare its type but no value. In this case, the variable will be set to undefined.

```
var name:string;

//The variable is a string variable. The variable’s value is set to
undefined by default
```

3. Declare its value but no type. The variable type will be set to the data type of the assigned value.

```
var name = ”mary”

//The variable’s type is inferred from the data type of the value. Here,
the variable is of the type string
```

4. Declare neither value not type. In this case, the data type of the variable will be any and will be initialized to undefined.

```
var name;

//The variable's data type is any. Its value is set to undefined by
default.
```

Does TypeScript support function overloading?

- TypeScript provides the concept of function overloading. You can have multiple functions with the same name but different parameter types and return type. However, the number of parameters should be the same.

```
function add(a:string,b:string) :string;

function add(a:number,b:number) :number;

function add(a:any, b:any) :any{
    return a + b
};

console.log(add("suraj","porje"));// surajporje
console.log(add(30,20)) //50
```

- Function overloading with different number of parameters and types with same name is not supported.
- Thus, in order to achieve function overloading, we must declare all the functions with possible signatures.
- Also, function implementation should have compatible types for all declarations.

Does TypeScript support all object oriented principles?give reason?

- TypeScript is object oriented JavaScript.
- TypeScript supports object-oriented programming features like classes, interfaces, etc.
- A class in terms of OOP is a blueprint for creating objects.
- A class encapsulates data for the object.
- Typescript gives built in support for this concept called class.
- JavaScript ES5 or earlier didn't support classes.
- Typescript gets this feature from ES6.
- Reasons

1. TypeScript supports the concept of Inheritance.
- A class inherits from another class using the 'extends' keyword. Child classes inherit all properties and methods except private members and constructors from the parent class.
- Multiple inheritence – (A class can inherit from multiple classes) TypeScript doesn't support multiple inheritance.

2. Encapsulation
- A class can control the visibility of its data members to members of other classes.
- access modifiers supported by TypeScript are –
    1. public
        - A public data member has universal accessibility. Data members in a class are public by default.
    2. private
        - Private data members are accessible only within the class that defines these members. If an external class member tries to access a private member, the compiler throws an error.
    - 3.protected
        - A protected data member is accessible by the members within the same class as that of the former and also by the members of the child classes.

3. Abstraction
- Abstraction is a way to model objects in a system that creates a separation of duties between class and the code that inherits it.
- It's also a way to enforce a concept called contract based development, A developer creates a class or interface, and that class type specifies what the calling code should implement, but not how.
    - So it's the job of the abstract type to define what needs to be done,
    - but up to the consuming types(object/consumer) to actually do those things.
- IN Typescript, To enforce abstraction, inherit from abstract classes/interfaces using extends/implement keyword

4. Polymorphism.
- When multiple classes inherit from a parent and override the same functionality, the result is polymorphism. Each of those child classes now implements a property or method, but they each may have their own way of performing that implementation.
- in typescript it can be achieved , by making the
    - method/property as abstract
    - and using the same method/property name in sub classes for that specific method implementation

What is getters/setters in TypeScript?

- The getters and setters allow you to control the access to the properties of a class.

1. A getter method
    - returns the value of the property's value.
    - A getter is also called an accessor.
    - A getter method starts with the keyword **get**

```
  // getter to get age
     public get age() {
         return this._age;
     }
```

2. A setter method
    - updates the property's value.
    - A setter is also known as a mutator.
    - a setter method starts with the keyword **set**

```
//setter to set age with check
 public set age(theAge: number) {
    if (theAge <= 0 || theAge >= 200) {
        throw new Error('The age is invalid');
    }
    this._age = theAge;
}
```

    - so setter can be used to avoid repeating the check

What is super in TypeScript?

- The super keyword is used to refer to the immediate parent of a class.
- It can be used in expressions to reference base class properties and the base class constructor or methods.
- Super calls consist of the keyword super followed by an argument list enclosed in parentheses.
- Super calls are only permitted in constructors of derived classes.

```
super( data members,...)
super.methods()
```

Which keyword is used for inheritance in TypeScript? Example?

- A class inherits from another class using the 'extends' keyword.
- Child classes inherit all properties and methods except private members and constructors from the parent class.
- e.g

```
class Shape{

    _Area: number
```

```
    constructor (Area:number)
    {
        this._Area = Area
    }
}

class Circle extends Shape {

   display() :void{
       console.log(`Area of circle is ${this._Area}`)
   }
}

const c1 = new Circle(100)

c1.display() //Area of circle is 100
```

Write a function that converts user entered date formatted as (MM/DD/YYYY) to a (YYYYMMDD) format.The parameter "userdate" and the return value are strings.

> For Example,it should convert user entered date "12/31/2019" to "20191231" function formatDate (userDate){ //write your code here } console.log (formatDate("12/31/2019"))

-

```
function formatDates(userDate:string)
{ // "12/31/2019"  outpot 2019,12,31
  const date = userDate.split('/')

  console.log(`date is ${date[2] + date[0] + date[1]}`)
     const result = date[2] + date[0] + date[1]
     return result
}

formatDates("12/31/2019")
//or
console.log(formatDates("12/31/2019"))
```

How to check type of variable in typescript?

- using typeof() fucntion

TypeScript Variable Scope

- The scope of a variable specifies where the variable is defined. The availability of a variable within a program is determined by its scope.
- TypeScript variables can be of the following scopes –

1. Global Scope – Global variables are declared outside the programming constructs. These variables can be accessed from anywhere within your code.
2. Class Scope – These variables are also called fields. Fields or class variables are declared within the class but outside the methods. These variables can be accessed using the object of the class. Fields can also be static. Static fields can be accessed using the class name.
3. Local Scope – Local variables, as the name suggests, are declared within the constructs like methods, loops etc. Local variables are accessible only within the construct where they are declared.

```
var global_num = 12          //global variable
class Numbers {
   num_val = 13;             //class variable
   static sval = 10;         //static field

   storeNum():void {
      var local_num = 14;    //local variable
   }
}
console.log("Global num: "+global_num)
console.log(Numbers.sval)   //static variable
var obj = new Numbers();
console.log("Global num: "+obj.num_val)
```

What do you understand by classes in Typescript? List some features of classes.

- TypeScript is object oriented JavaScript.
- TypeScript supports object-oriented programming features like classes, interfaces, etc.
- A class in terms of OOP is a blueprint for creating objects.
- A class encapsulates data for the object.
- Typescript gives built in support for this concept called class.
- JavaScript ES5 or earlier didn't support classes.
- Typescript gets this feature from ES6.
- classes are the fundamental entities used to create reusable components.
- Functionalities are passed down to classes and objects are created from classes.
- The class in TypeScript is compiled to plain JavaScript functions by the TypeScript compiler to work across platforms and browsers.
- features :

1. **Constructor**

- The constructor is a special type of method which is called when creating an object. In TypeScript, the constructor method is always defined with the name "constructor".

It is not necessary for a class to have a constructor.

2. **Creating an Object of Class**

- An object of the class can be created using the new keyword.
- If the class includes a parameterized constructor, then we can pass the values while creating the object.

3. **Inheritance** Just like object-oriented languages such as Java and C#, TypeScript classes can be extended to create new classes with inheritance, using the keyword extends.

- We must call super() method first before assigning values to properties in the constructor of the derived class.

4. **Class Implements Interface**

5. class can implement **single or multiple interfaces**.

6. **Interface extends Class** An interface can also extend a class to represent a type.

Example: Interface Extends Class Copy

```
class Person {
    name: string;
}

interface IEmployee extends Person {
    empCode: number;
}

let emp: IEmployee = { empCode  : 1, name:"James Bond" }
```

In the above example, IEmployee is an interface that extends the Person class. So, we can declare a variable of type IEmployee with two properties. So now, we must declare and initialize values at the same time.

7. **Method Overriding**

- When a child class defines its own implementation of a method from the parent class, it is called method overriding.

What is needs to done if i want option A as output(note: dont change 1st line of code)

```
 let a:string=47;

console.log( " Value of a= " +a);
A) Value of a= 47
B) Value of a= 0
```

```
C) Value of a=
D) Error
```

- console.log(value of a= ${a});

find the bug

```
function average(a, b) {
var z=parseInt(a)
return z + b/ 2;
}

var num1:string="10"
var num2:number=30
console.log(average(num1, num2)); // "10" + 30/2

Find the bug in given code
```

- result 25, no error
- return (z+b)/2, bug

A _____ data member is accessible by the members within the same class as that of the former and also by the members of the child classes.

- A. protected
- B. public
- C. private
- D. All of the above
- Ans : A

day 3 _ sept 16

Explain angular

- Angular is a TypeScript-based open-source web application framework, developed and maintained by Google.
- It offers an easy and powerful way of building front end web-based applications.
- Angular integrates a range of features like declarative templates, dependency injection, end-to-end tooling, etc. that facilitates web application development.
- Angular is a platform and framework for building single-page client applications using HTML and TypeScript.
- Angular is written in TypeScript.
- It implements core and optional functionality as a set of TypeScript libraries that you import into your apps.
- The basic building blocks of angular are NgModules,
- an Angular app is defined by a set of NgModules.

    - An app always has at least a root module that enables bootstrapping, and typically has many more feature modules.

What are the advantages of angular?

- The advantages of AngularJS are –
    - It provides the capability to create Single Page Application in a very clean and maintainable way.
    - It provides data binding capability to HTML. Thus, it gives user a rich and responsive experience.- AngularJS code is unit testable.
    - AngularJS uses dependency injection and make use of separation of concerns.
    - AngularJS provides reusable components.
    - With AngularJS, the developers can achieve more functionality with short code.
    - In AngularJS, views are pure html pages, and controllers written in JavaScript do the business processing.
    - On the top of everything, AngularJS applications can run on all major browsers and smart phones, including Android and iOS based phones/tablets.
- Disadvantages of AngularJS
- Though AngularJS comes with a lot of merits, here are some points of concern –
    - Not Secure – Being JavaScript only framework, application written in AngularJS are not safe. Server side authentication and authorization is must to keep an application secure.
    - Not degradable – If the user of your application disables JavaScript, then nothing would be visible, except the basic page.

purpose of angular assets folder?

- Contains image and other asset files to be copied as-is when you build your application.
- Assets folder is used in angular for maintaining file which doesn't have to be modified while compiling.
- You can't maintain any files like Images, json or any other file types in a component folder(folders are only for development purpose only) because it will be not recognized while compiling.
- If you are in any component always set image path like this. Always maintain assets as a starting folder

```
assets/pic.img
```

what exactly does ng new in angular?

- Creates a new workspace and an initial Angular app.

```
ng new <name> [options]
```

```
ng n <name> [options]
```

- Creates and initializes a new Angular app that is the default project for a new workspace.
- Provides interactive prompts for optional configuration, such as adding routing support. All prompts can safely be allowed to default.
- The new workspace folder is given the specified project name, and contains configuration files at the top level.
- By default, the files for a new initial app (with the same name as the workspace) are placed in the src/ subfolder. Corresponding end-to-end tests are placed in the e2e/ subfolder.
- The new app's configuration appears in the projects section of the angular.json workspace configuration file, under its project name.

what exactly does ng serve in angular?

- ng serve command builds and serve the application.
- It rebuilds the application if changes occur.
- Syntax

```
ng serve <project> [options]
ng s <project> [options]
//project The name of the project to build. Can be an application or a
library.
```

- Options are optional parameters.
- e.g
    1. --host=host
        - Host to listen on.Default: localhost
    2. --port
        - Port to listen on.Default: 4200
    3. --watch=true|false
        - Rebuild on change.Default: true
    4. --open=true|false or -o= true|false
        - Opens the url in default browser.Default: false
        - Aliases: -o
    5. --help=true|false|json|JSON
        - Shows a help message for this command in the console.
        - Default: false

What is angular JSON and package JSON?

1. A file named angular.json
    - at the root level of an Angular workspace provides workspace-wide and project-specific configuration defaults for build and development tools provided by the Angular CLI.

2. package.json

- Both npm and yarn install the packages that are identified in a package.json file.
- The CLI command ng new creates a package.json file when it creates the new workspace.
    - This package.json is used by all projects in the workspace,
    - including the initial app project that is created by the CLI when it creates the workspace.
    - Initially, this package.json includes a starter set of packages, some of which are required by Angular and others that support common application scenarios.
    - You add packages to package.json as your application evolves. You may even remove some.
    - The package.json is organized into two groups of packages:
    - Dependencies are essential to running applications.
    - DevDependencies are only necessary to develop applications.

What is a single page application in AngularJS?

- AngularJS is one of the first JS frameworks for single page applications. It is a set of tools that significantly simplifies SPA development.
- SPA means Single Page Applications
- Such web applications load the HTML, CSS or JS of the page once, and don't require reloading within page-to-page navigation.
- three main SPA advantages.

1. Speed
    - On SPA, the page navigation seems to be quick enough thank to caching process. It means that an application will not load the whole page at the request of a user, it will only download the required parts.
2. Caching process
    - On the one hand, an application never stops checking if the previously downloaded data remains the same to update it when needed. So, the app doesn't waste time loading the whole page again and again. - On the other hand, the Multiple Page Application successfully uses caching as well, though the download speed here depends more on business needs and can be increased if necessary.

3.UX

- UX design in SPA is generally better and more pleasant for an end-user.
- A great SPA advantage and the main reason for turning to SPA is its high interactivity an d possibility to work offline.
- Thus, it seems that you're dealing with a desktop application rather than with a web page.
- So, Single Page Application built in AngularJS is quick to build and easy to use, though it possesses some cons as well as pros.

NgModel DIRECTIVE

- Creates a FormControl instance from a domain model and binds it to a form control element.
- Description
- The **FormControl** instance tracks the value, user interaction, and validation status of the control and keeps the view synced with the model. If used within a parent form, the directive also registers itself with the form as a child control.
- This directive is used by itself or as part of a larger form. Use the ngModel selector to activate it.
- It accepts a domain model as an optional Input.

1. If you have a one-way binding to ngModel with [] syntax, changing the value of the domain model in the component class sets the value in the view.
2. If you have a two-way binding with [()] syntax (also known as 'banana-box syntax'), the value in the UI always syncs back to the domain model in your class.

To inspect the properties of the associated FormControl (like validity state), export the directive into a local template variable using ngModel as the key (ex: $myVar="ngModel").

- When using the ngModel within

  tags, you'll also need to supply a name attribute so that the control can be registered with the parent form under that name.
- In the context of a parent form, it's often unnecessary to include one-way or two-way binding, as the parent form syncs the value for you.
- You access its properties by exporting it into a local template variable using ngForm such as (#f="ngForm"). Use the variable where needed on form submission.

Using ngModel within a form The following example shows controls using ngModel within a form:

```
import {Component} from '@angular/core';
import {NgForm} from '@angular/forms';

@Component({
  selector: 'example-app',
  template: `
    <form #f="ngForm" (ngSubmit)="onSubmit(f)" novalidate>
      <input name="first" ngModel required #first="ngModel">
      <input name="last" ngModel>
      <button>Submit</button>
    </form>

    <p>First name value: {{ first.value }}</p>
    <p>First name valid: {{ first.valid }}</p>
    <p>Form value: {{ f.value | json }}</p>
    <p>Form valid: {{ f.valid }}</p>
  `,
```

```
})
export class SimpleFormComp {
  onSubmit(f: NgForm) {
    console.log(f.value);  // { first: '', last: '' }
    console.log(f.valid);  // false
  }
}
```

HttpClient performs

- A. HTTP requests.
- B. This service is available as an injectable class, with methods to perform HTTP requests.
- C. Each request method has multiple signatures, and the return type varies based on the signature that is called .
- D.all of the above
- Ans . D

mcq

- Q.Most front-end applications need to communicate with a server over the HTTP protocol, in order to download or upload data and access other back-end services.
- Angular provides a simplified client HTTP API for Angular applications, the HttpClient service class in @angular/common/http. The HTTP client service offers which of the following features:
- A. The ability to request typed response objects.
- B. Streamlined error handling.
- C. Testability features.
- D. Request and response interception.
- Ans : all of the above

Dependency injection (DI), are

- A. an important application design pattern.
- B. Angular own DI framework, is typically used in the design of Angular applications to increase their efficiency and modularity.
- C. Dependencies are services or objects that a class needs to perform its function.
- D. it is a coding pattern in which a class asks for dependencies from external sources rather than creating them itself.
- E. In Angular, the DI framework provides declared dependencies to a class when that class is instantiated.
- F. use to make your apps flexible, efficient, and robust, as well as testable and maintainable.

- G. All of the above
- Ans: G