TypeScript Interview Questions

A list of top frequently asked **TypeScript Interview Questions** and answers are given below.

1) What is Typescript?

TypeScript is a free and open-source programming language developed and maintained by **Microsoft.** It is a strongly typed superset of JavaScript that compiles to plain JavaScript. It is a language for application-scale JavaScript development. TypeScript is quite easy to learn and use for developers familiar with C#, Java and all strong typed languages.

TypeScript can be executed on Any browser, Any Host, and Any Operating System. TypeScript is not directly run on the browser. It needs a compiler to compile and generate in JavaScript file. TypeScript is the **ES6** version of JavaScript with some additional features.

2) How is TypeScript different from JavaScript?

TypeScript is different from JavaScript in the following manner:

SN	JavaScript	TypeScript
1	It was developed by Netscape in 1995.	It was developed by Anders Hejlsberg in 2012.
2	JavaScript source file is in ".js" extension.	TypeScript source file is in ".ts" extension.

3	JavaScript doesn't support ES6.	TypeScript supports ES6.
4	It doesn't support strongly typed or static typing.	It supports strongly typed or static typing feature.
5	It is just a scripting language.	It supports object-oriented programming concept like classes, interfaces, inheritance, generics, etc.
6	JavaScript has no optional parameter feature.	TypeScript has optional parameter feature.
7	It is interpreted language that's why it highlighted the errors at runtime.	It compiles the code and highlighted errors during the development time.
8	JavaScript doesn't support modules.	TypeScript gives support for modules.
9	In this, number, string are the objects.	In this, number, string are the interface.
10	JavaScript doesn't support generics.	TypeScript supports generics.

To know more click here.

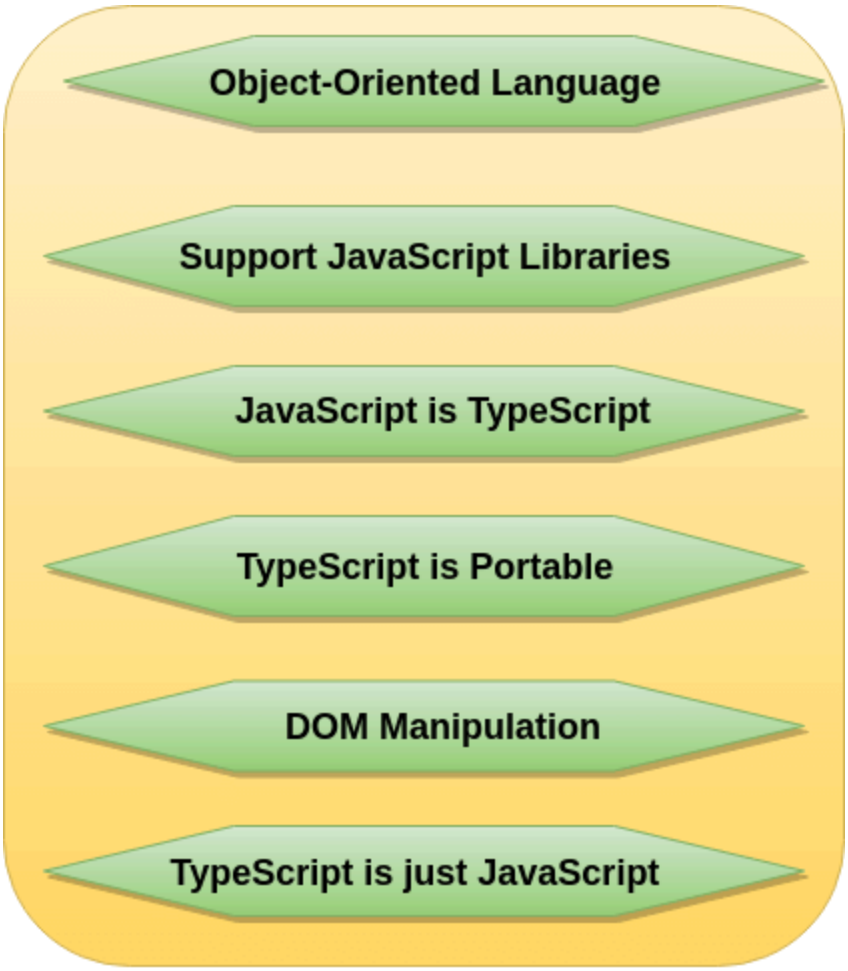
3) Why do we need TypeScript?

We need TypeScript:

- TypeScript is fast, simple, and most importantly, easy to learn.

- TypeScript supports object-oriented programming features such as classes, interfaces, inheritance, generics, etc.
- TypeScript provides the error-checking feature at compilation time. It will compile the code, and if any error found, then it highlighted the errors before the script is run.
- TypeScript supports all JavaScript libraries because it is the superset of JavaScript.
- TypeScript support reusability by using the inheritance.
- TypeScript make app development quick and easy as possible, and the tooling support of TypeScript gives us autocompletion, type checking, and source documentation.
- TypeScript supports the latest JavaScript features including ECMAScript 2015.
- TypeScript gives all the benefits of ES6 plus more productivity.
- TypeScript supports Static typing, Strongly type, Modules, Optional Parameters, etc.

4) List some features of Typescript?



Features of TypeScript

To know more click here.

5) List some benefits of using Typescript?

TypeScript has the following benefits.

- It provides the benefits of optional static typing. Here, Typescript provides types that can be added to variables, functions, properties, etc.
- Typescript has the ability to compile down to a version of JavaScript that runs on all browsers.
- TypeScript always highlights errors at compilation time during the time of development whereas JavaScript points out errors at the runtime.
- TypeScript supports strongly typed or static typing whereas this is not in JavaScript.
- It helps in code structuring.

- It uses class-based object-oriented programming.
- It provides excellent tooling supports with IntelliSense which provides active hints as the code is added.
- It has a namespace concept by defining a module.

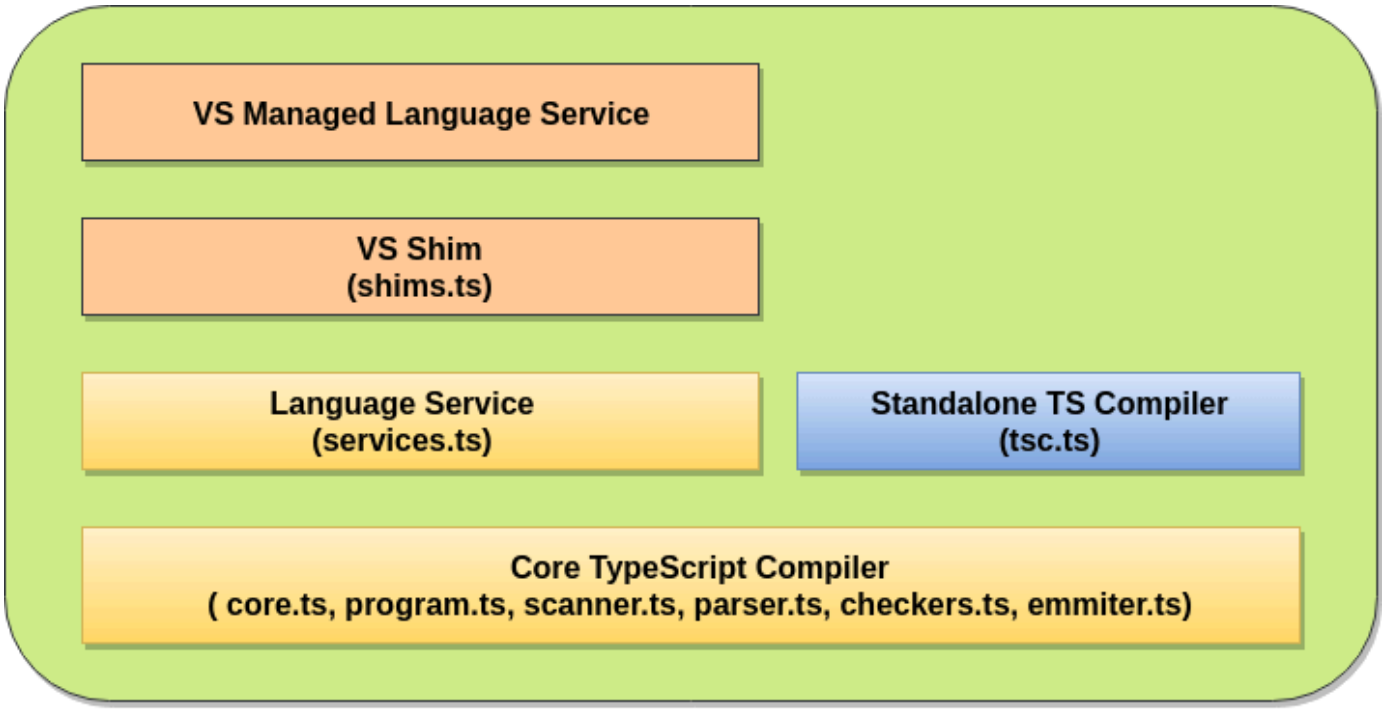
6) What are the disadvantages of TypeScript?

TypeScript has the following disadvantages:

- TypeScript takes a long time to compile the code.
- TypeScript does not support abstract classes.
- If we run the TypeScript application in the browser, a compilation step is required to transform TypeScript into JavaScript.
- Web developers are using JavaScript from decades and TypeScript doesn?t bring anything new.
- To use any third party library, the definition file is must. And not all the third party library have definition file available.
- Quality of type definition files is a concern as for how can you be sure the definitions are correct?

7) What are the different components of TypeScript?

The TypeScript has mainly three components. These are-



Components of TypeScript

Language

The language comprises elements like new syntax, keywords, type annotations, and allows us to write TypeScript.

Compiler

The TypeScript compiler is open source, cross-platform, and is written in TypeScript. It transforms the code written in TypeScript equivalent to its JavaScript code. It performs the parsing, type checking of our TypeScript code to JavaScript code. It can also help in concatenating different files to the single output file and in generating source maps.

Language Service

The language service provides information which helps editors and other tools to give better assistance features such as automated refactoring and IntelliSense.

To know more click here.

8) Who developed Typescript and what is the current stable version of Typescript?

The typescript was developed by **Anders Hejlsberg**, who is also one of the core members of the development team of C# language. The typescript was first released in the month of October **1st, 2012** and was labeled version **0.8**. It is developed and maintained by Microsoft under the **Apache 2** license. It was designed for the development of a large application.

The current stable version of TypeScript is **3.2** which was released on September 30, 2018. Typescript compiles to simple JavaScript code which runs on any browser that supports ECMAScript 2015 framework. It offers support for the latest and evolving JavaScript features.

9) Tell the minimum requirements for installing Typescript. OR how can we get TypeScript and install it?

TypeScript can be installed and managed with the help of node via npm (the Node.js package manager). To install TypeScript, first ensure that the **npm** is installed correctly, then run the following command which installs TypeScript globally on the system.

```
$ npm install -g typescript
```

It installs a command line code **"tsc"** which will further be used to compile our Typescript code. Make sure that we check the version of Typescript installed on the system.

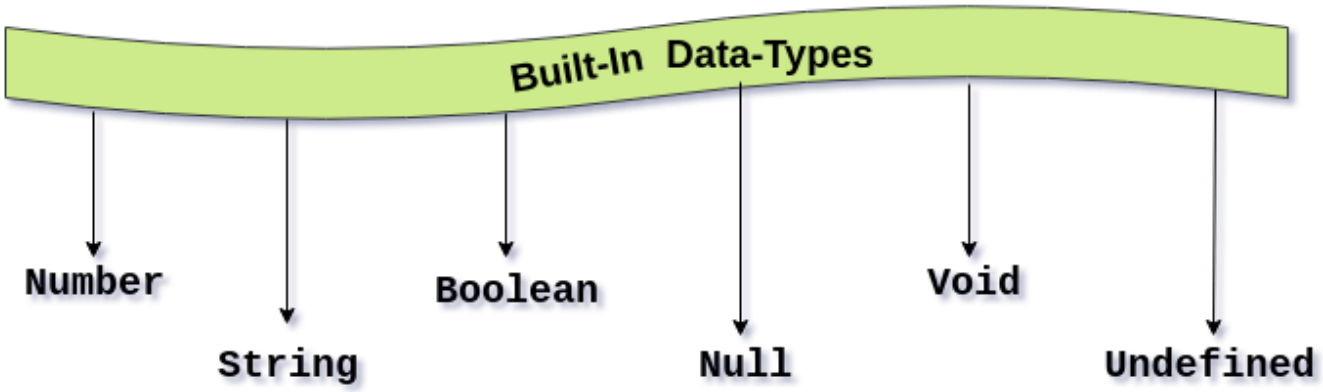
Following steps are involved for installing TypeScript:

1. Download and run the .msi installer for the node.
2. Enter the command "node -v" to check if the installation was successful.
3. Type the following command in the terminal window to install Typescript: $ npm install -g typescript

To know installation process click here.

10) List the built-in types in Typescript.

The built-in data types are also known as primitive data types in Typescript. These are given below.



Number type: It is used to represent number type values. All the numbers in TypeScript are stored as floating point values.

Syntax: let identifier: number = value;

String type: It represents a sequence of characters stored as Unicode UTF-16 code. We include string literals in our scripts by enclosing them in single or double quotation marks.

Syntax: let identifier: string = " ";

Boolean type: It is used to represent a logical value. When we use the Boolean type, we get output only in true or false. A Boolean value is a truth value that specifies whether the condition is true or not.

Syntax: let identifier: bool = Boolean value;

Null type: Null represents a variable whose value is undefined. It is not possible to directly reference the null type value itself. Null type is not useful because we can only assign a null value to it.

Syntax: let num: number = null;

Undefined type: It is the type of undefined literal. The Undefined type denotes all uninitialized variables. It is not useful because we can only assign an undefined value to it. This type of built-in type is the sub-type of all the types.

Syntax: let num: number = undefined;

Void type: A void is the return type of the functions that do not return any type of value. It is used where no datatype is available.

Syntax: let unusable: void = undefined;

To know TypeScript datatypes in detail click here.

11) What are the variables in Typescript? How to create a variable in Typescript?

A variable is the storage location, which is used to store value/information to be referenced and used by programs. It acts as a container for value in a program. It can be declared using the var keyword. It should be declared before the use. While declaring a variable in Typescript, certain rules should be followed-

- The variable name must be an alphabet or numeric digits.
- The variable name cannot start with digits.
- The variable name cannot contain spaces and special character, except the underscore(_) and the dollar($) sign.

We can declare a variable in one of the four ways:

1. Declare type and value in a single statement. Syntax: var [identifier] : [type-annotation] = value;
2. Declare type without value. Syntax: var [identifier] : [type-annotation];
3. Declare its value without type. Syntax: var [identifier] = value;
4. Declare without value and type. Syntax: var [identifier];

To know more in detail click here.**https://www.javatpoint.com/typescript-variables**

12) How to compile a Typescript file?

Here is the command which is followed while compiling a Typescript file into JavaScript.

```
$ tsc <TypeScript File Name>
```

For example, to compile "Helloworld.ts."

```
$ tsc helloworld.ts
```

The result would be **helloworld.js.**

13) Is it possible to combine multiple .ts files into a single .js file? If yes, then how?

Yes, it is possible. For this, we need to add **--outFILE** [OutputJSFileName] compiling option.

```
$ tsc --outFile comman.js file1.ts file2.ts file3.ts
```

The above command will compile all three **".ts"**file and result will be stored into single **"comman.js"** file. In the case, when we don't provide an output file name as like in below command.

```
$ tsc --outFile file1.ts file2.ts file3.ts
```

Then, the **file2.ts** and **file3.ts** will be compiled, and the output will be placed in **file1.ts.** So now our file1.ts contains JavaScript code.

14) Is it possible to compile .ts automatically with real-time changes in the .ts file?

Yes, it is possible to compile ".ts" automatically with real-time changes in the .ts file. This can be achieved by using --watch compiler option

```
tsc --watch file1.ts
```

The above command first compiles **file1.ts** in **file1.js** and watch for the file changes. If there is any change detected, it will compile the file again. Here, we need to ensure that command prompt must not be closed on running with **--watch** option.

15) What do you mean by interfaces? Explain them with reference to Typescript.

An Interface is a structure which acts as a contract in our application. It defines the syntax for classes to follow, it means a class that implements an interface is bound to implement all its members. It cannot be instantiated but can be referenced by the class object that implements it. The TypeScript compiler uses interface for type-checking (also known as "duck typing" or "structural subtyping") whether the object has a specific structure or not.

Syntax:

```
interface interface_name {
        // variables' declaration
        // methods' declaration
}
```

The interface just declares the methods and fields. It cannot be used to build anything. Interfaces need not be converted to JavaScript for execution. They have zero runtime JavaScript impact. Thus, their only purpose is to help in the development stage.

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

We know, TypeScript is a type of Object-Oriented JavaScript language and supports OOPs programming features like classes, interfaces, etc. Like Java, classes are the fundamental entities which are used to create reusable components. It is a group of objects which have common properties. A class is a template or blueprint for creating objects. It is a logical entity. The "class" keyword is used to declare a class in Typescript.

Example:

```
class Student {
   studCode: number;
   studName: string;
   constructor(code: number, name: string) {
         this.studName = name;
         this.studCode = code;
   }
   getGrade() : string {
      return "A+" ;
   }
```

```
}
```

Features of a class are-

1. Inheritance
2. Encapsulation
3. Polymorphism
4. Abstraction

17) Is Native Javascript supports modules?

No. Currently, modules are not supported by Native JavaScript. To create and work with modules in Javascript we require an external like CommonJS.

18) Which object oriented terms are supported by TypeScript?

TypeScript supports following object oriented terms.

- Modules
- Classes
- Interfaces
- Inheritance
- Data Types
- Member functions

19) How to Call Base Class Constructor from Child Class in TypeScript?

super() function is used to called parent or base class constructor from Child Class.

20) How do you implement inheritance in TypeScript?

Inheritance is a mechanism that acquires the properties and behaviors of a class from another class. It is an important aspect of OOPs languages and has the ability which creates

new classes from an existing class. The class whose members are inherited is called the base class, and the class that inherits those members is called the derived class.

An Inheritance can be implemented by using the extend keyword. We can understand it by the following example.

```
class Shape {
  Area:number
  constructor(area:number) {
    this.Area = area
  }
}
class Circle extends Shape {
  display():void {
    console.log("Area of the circle: "+this.Area)
  }
}
var obj = new Circle(320);
obj.display()  //Output: Area of the circle: 320
```

To know more click here.

21) What are the Modules in Typescript?

A module is a powerful way to create a group of related variables, functions, classes, and interfaces, etc. It can be executed within their own scope, not in the global scope. In other words, the variables, functions, classes, and interfaces declared in a module cannot be accessible outside the module directly.

Creating a Module

A module can be created by using the **export** keyword and can be used in other modules by using the **import** keyword.

```
module module_name{
   class xyz{
      export sum(x, y){
         return x+y;
```

```
        }
    }
}
```

To know more click here.

22) What is the difference between the internal module and the external module?

The difference between internal and external module is given below:

SN	Internal Module	External Module
1	Internal modules were used to logically group the classes, interfaces, functions, variables into a single unit and can be exported in another module.	External modules are useful in hiding the internal statements of the module definitions and show only the methods and parameters associated with the declared variable.
2	Internal modules were in the earlier version of Typescript. But they are still supported by using namespace in the latest version of TypeScript.	External modules are simply known as a module in the latest version of TypeScript.
3	Internal modules are local or exported members of other modules (including the global module and external modules).	External modules are separately loaded bodies of code referenced using external module names.
4	Internal modules are declared using ModuleDeclarations that	An external module is written as a separate source file that contains

	specify their name and body.	at least one import or export declaration.
5	Example: `module Sum { export function add(a, b) { console.log("Sum: " +(a+b)); } }`	Example: `export class Addition{ constructor(private x?: number, private y?: number){ } Sum(){ console.log("SUM: " +(this.x + this.y)); } }`

To know more in detail click here.

23) What is namespace in Typescript? How to declare a namespace in Typescript?

A namespace is a way that is used for logical grouping of functionalities. Namespaces are used to maintain the legacy code of typescript internally. It encapsulates the features and objects that share certain relationships. A namespace is also known as internal modules. A namespace can also include interfaces, classes, functions, and variables to support a group of related functionalities.

Note: A namespace can be defined in multiple files and allow to keep each file as they were all defined in one place. It makes code easier to maintain.

Synatax for creating namespace

```
namespace <namespace_name> {
    export interface I1 { }
```

```
        export class c1{ }
}
```

To know more click here.

24) Explain Decorators in Typescript?

A Decorator is a special kind of declaration that can be applied to classes, methods, accessor, property, or parameter. Decorators are simply functions that are prefixed **@expression** symbol, where expression must evaluate to a function that will be called at runtime with information about the decorated declaration.

TypeScript Decorators serves the purpose of adding both annotations and metadata to the existing code in a declarative way. Decorators are an experimental feature proposed for **ES7.** It is already in use by some of the JavaScript frameworks including **Angular 2.** The Decorators may change in future releases.

To enable experimental support for decorators, we must enable the experimentalDecorators compiler option either on the command line or in our tsconfig.json:

Command Line

```
$tsc --target ES5 --experimentalDecorators
```

tsconfig.json

```
{
   "compilerOptions": {
      "target": "ES5",
      "experimentalDecorators": true
   }
}
```

To know more click here.

25) What are Mixins?

In Javascript, Mixins are a way of building up classes from reusable components is to build them by combining simpler partial classes called mixins.

The idea is simple, instead of a class A extending class B to get its functionality, function B takes class A and returns a new class with this added functionality. Function B is a mixin.

26) What is default visibility for properties/methods in TypeScript classes?

Public is the default visibility for properties/methods in TypeScript classes.

27) How does TypeScript support optional parameters in function as in JavaScript every parameter is optional for a function?

Unlike JavaScript, the TypeScript compiler will throw an error if we try to invoke a function without providing the exact number and types of parameters as declared in its function signature. To overcome this problem, we can use optional parameters by using question mark sign ('?'). It means that the parameters which may or may not receive a value can be appended with a '?' to mark them optional.

```
function Demo(arg1: number, arg2? :number) {
}So, arg1 is always required, and arg2 is an optional parameter.
```

So, **arg1** is always required, and **arg2** is an optional parameter.

Note: Optional parameters must follow the required parameters. If we want to make **arg1** optional, instead of **arg2,** then we need to change the order and arg1 must be put after arg2.

```
function Demo(arg2: number, arg1? :number) {
}
```

To know more click here.

28) Does TypeScript supports function overloading as JavaScript doesn't support function overloading?

Yes, TypeScript support function overloading. But the implementation is odd. When we perform function overloading in TypeScript, then we can implement only one functions

with multiple signatures.

```
//Function with string type parameter
function add(a:string, b:string): string;

//Function with number type parameter
function add(a:number, b:number): number;

//Function Definition
function add(a: any, b:any): any {
   return a + b;
}
```

In the above example, the first two lines are the function overload declaration. It has two overloads. The first signature has a parameter of type string whereas the second signature has a parameter of type number. The third function contains the actual implementation and has a parameter of type any. Any data type can take any type of data. The implementation then checks for the type of the supplied parameter and execute a different piece of code based on supplier parameter type.

29) Is it possible to debug any TypeScript file?

Yes, it is possible. To debug any TypeScript file, we need .js source map file. So compile the .ts file with the **--sourcemap** flag to generate a source map file.

```
$ tsc -sourcemap file1.ts
```

This will create **file1.js** and **file1.js.map.** And last line of **file1.js** would be reference of source map file.

```
//# sourceMappingURL=file1.js.map
```

30) What is TypeScript Definition Manager and why do we need it?

TypeScript Definition Manager (TSD) is a package manager used to search and install TypeScript definition files directly from the community-driven DefinitelyTyped repository.

Suppose, we want to use some jQuery code in our .ts file.

```
$(document).ready(function() { //Your jQuery code });
```

Now, when we try to compile it by using **tsc,** it will give a compile-time error: Cannot find the name **"$".** So, we need to inform TypeScript compiler that **"$"** is belongs to jQuery. To do this, TSD comes into play. We can download **jQuery** Type Definition file and include it in our **.ts** file. Below are the steps to perform this:

First, install TSD.

```
$ npm install tsd -g
```

In TypeScript directory, create a new TypeScript project by running:

```
$ tsd init
```

Then install the definition file for jQuery.

```
tsd query jquery --action install
```

The above command will download and create a new directory containing jQuery definition file ends with ".d.ts". Now, include definition file by updating TypeScript file to point to the jQuery definition.

```
/// <reference path="typings/jquery/jquery.d.ts" />
$(document).ready(function() { //To Do
});
```

Now, compile again. This time js file will be generated without any error. Hence, the need of TSD helps us to get type definition file for the required framework.

31) What is TypeScript Declare Keyword?

We know that all JavaScript libraries/frameworks don't have TypeScript declaration files, but we want to use them in our TypeScript file without any compilation errors. To do this, we use the declare keyword. The declare keyword is used for ambient declarations and methods where we want to define a variable that may exist elsewhere.

For example, suppose we have a library called myLibrary that doesn't have a TypeScript declaration file and have a namespace called myLibrary in the global namespace. If we want to use that library in our TypeScript code, we can use the following code:

```
declare var myLibrary;
```

TypeScript runtime will assign the myLibrary variable as any type. Here is a problem that we won't get Intellisense in design time but we will be able to use the library in our code.

32) How to generate TypeScript definition file from any .ts file?

We can generate TypeScript definition file from any .ts file by using tsc compiler. It will be generating a TypeScript definition which makes our TypeScript file reusable.

```
tsc --declaration file1.ts
```

33) What is tsconfig.json file?

The tsconfig.json file is a file which is in JSON format. In the tsconfig.json file, we can specify various options to tell the compiler how to compile the current project. The presence of a tsconfig.json file in a directory indicates that the directory is the root of a TypeScript project. Below is a sample tsconfig.json file.

```
{
  "compilerOptions": {
    "declaration": true,
    "emitDecoratorMetadata": false,
    "experimentalDecorators": false,
    "module": "none",
    "moduleResolution": "node"
```

```
    "removeComments": true,
    "sourceMap": true
  },
  "files": [
    "main.ts",
    "othermodule.ts"
  ]
}
```

To know more click here.

34) Explain generics in TypeScript?

TypeScript Generics is a tool which provides a way to create reusable components. It is able to create components that can work with a variety of data types rather than a single data type. Generics provides type safety without compromising the performance, or productivity. Generics allow us to create generic classes, generic functions, generic methods, and generic interfaces.

In generics, a type parameter is written between the open (<) and close (>) brackets which makes it strongly typed collections. Generics use a special kind of type variable <T> that denotes types. The generics collections contain only similar types of objects.

```
function identity<T>(arg: T): T {
   return arg;
}
let output1 = identity<string>("myString");
let output2 = identity<number>( 100 );
console.log(output1);
console.log(output2);
```

To know more click here.

35) Does TypeScript support all object-oriented principles?

Yes, TypeScript support all object-oriented principles. There are four main principles to object-oriented programming:

1. Encapsulation,
2. Inheritance,
3. Abstraction, and
4. Polymorphism.

36) How to check null and undefined in TypeScript?

By using a juggling-check, we can check both null and undefined:

```
if (x == null) {
}
```

If we use a strict-check, it will always true for values set to null and won't evaluate as true for undefined variables.

Example

```
var a: number;
var b: number = null;
function check(x, name) {
    if (x == null) {
        console.log(name + ' == null');
    }
    if (x === null) {
        console.log(name + ' === null');
    }
    if (typeof x === 'undefined') {
        console.log(name + ' is undefined');
    }
}
check(a, 'a');
check(b, 'b');
```

Output

```
"a == null"
"a is undefined"
```

```
"b == null"
"b === null"
```

37) Could we use TypeScript on the backend? If yes, how?

Yes, we can use TypeScript on the backend. We can understand it with the following example. Here, we choose Node.js and have some additional type safety and the other abstraction that the language brings.

- Install Typescript compiler

```
npm i -g typescript
```

- The TypeScript compiler takes options in the tsconfig.json file. This file determines where to put built files.

```
{
  "compilerOptions": {
    "target": "es5",
    "module": "commonjs",
    "declaration": true,
    "outDir": "build"
  }
}
```

- Compile ts files

```
tsc
```

- Run

```
node build/index.js
```

38) What is the difference between "interface vs type" statements?

```
interface X {
    a: number
    b: string
}
type X = {
    a: number
    b: string
};
```

SN	interface	type
1	An interface declaration always introduces a named object type.	A type alias declaration can introduce a name for any kind of type, including primitive, union, and intersection types.
2	An interface can be named in an extends or implements clause.	Type alias for an object type literal cannot be named in an extends or implements clause.
3	Interfaces create a new name that is used everywhere.	Type aliases don't create a new name.
4	An interface can have multiple merged declarations.	Type alias for an object type literal cannot have multiple merged declarations.

39) What are Ambients in TypeScripts and when to use them?

Ambient declarations tell the compiler about the actual source code exist elsewhere. If these source codes do not exist at runtime and we try to use them, then it will break

without warning.

Ambient declarations files are like docs file. If the source changes, the docs need to be kept updated also. If the ambient declaration file is not updated, then we will get compiler errors.

The Ambient declarations allow us to safely and easily use existing popular JavaScript libraries like jquery, angularjs, nodejs, etc.

40) What is a TypeScript Map file?

- TypeScript Map file is a source map file which holds information about our original files.
- .map files are source map files that let tools map between the emitted JavaScript code and the TypeScript source files that created it.
- Many debuggers can consume these files so we can debug the TypeScript file instead of the JavaScript file.

41) What is Type assertions in TypeScript?

Type assertion works like a **typecasting** in other languages, but it doesn't perform **type checking** or **restructuring** of data just like other languages can do like C# and Java. The typecasting comes with runtime support whereas type assertion has no impact on **runtime.** However, type assertions are used purely by the compiler and provide hints to the compiler on how we want our code to be analyzed.

Example

```
let empCode: any = 111;
let employeeCode = <number> code;
console.log(typeof(employeeCode)); //Output: number
```

To know more click here.

42) What is "as" syntax in TypeScript?

The as is the additional syntax for Type assertion in TypeScript. The reason for introducing the as-syntax is that the original syntax (<type>) conflicted with JSX.

Example

```
let empCode: any = 111;
let employeeCode = code as number;
```

When using TypeScript with JSX, only as-style assertions are allowed.

43) What is JSX? Can we use JSX in TypeScript?

JSX is NOTHING BUT Javascript with a different extension. **Facebook** came up with this new extension so that they can distinguish from the XML-like implementation of HTML in JavaScript.

JSX is an embeddable **XML-like syntax.** It is meant to be transformed into valid JavaScript. JSX came to popularity with the React **framework.** TypeScript supports embedding, type checking, and compiling JSX directly into JavaScript.

To use JSX, we must do two things.

- Name the files with a **.tsx** extension
- Enable the **jsx** option

44) What is Rest parameters?

The rest parameter is used to pass **zero or more** values to a function. It is declared by prefixing the three **dot** characters **('...')**before the parameter. It allows the functions to have a variable number of arguments without using the arguments object. It is very useful where we have an undetermined number of parameters.

Rules to follow in rest parameter:

- Only one rest parameter is allowed in a function.
- It must be an array type.
- It must be a last parameter in the parameter list.

```
function sum(a: number, ...b: number[]): number {
 let result = a;
```

```
 for (var i = 0; i < b.length; i++) {
 result += b[i];
 }
 console.log(result);
}
let result1 = sum(3, 5);
let result2 = sum(3, 5, 7, 9);
```

To know more click here.

45) Explain Enum in TypeScript?

Enums or enumerations are a TypeScipt data type that allow us to define a set of named constants. Using enums can make it easier to document intent, or create a set of distinct cases. It is a collection of related values that can be numeric or string values.

Example

```
enum Gender {
  Male,
  Female
  Other
}
console.log(Gender.Female); // Output: 1
//We can also access an enum value by it's number value.
console.log(Gender[1]); // Output: Female
```

46) Explain Relative vs. Non-relative module imports.

Non-Relative	Relative
A non-relative import can be resolved relative to baseUrl, or through path mapping. In other words, we use non-relative paths when importing any of our external dependencies. **Example:**	Relative imports can be used for our own modules that are guaranteed to maintain their relative location at runtime. A relative import is starts with /, ./ or ../. **Example:**

```
import * as $ from "jquery";
import { Component } from
"@angular/core";
```

```
import Entry from
"./components/Entry";
import {DefaultHeaders} from
"../constants/http";
```

47) What is an anonymous function?

An anonymous function is a function that was declared without any **named identifier.** These functions are dynamically declared at runtime. Anonymous functions can accept inputs and return outputs, just as standard functions do. An anonymous function is usually not accessible after its initial creation.

Example

```
let myAdd = function(x: number, y: number): number {
return x + y;
};
console.log(myAdd())
```

48) What is Declaration Merging?

Declaration merging is the process followed by the compiler to merge **two** or **more** separate declarations. The declaration declared with the same name into a single definition. This merged definition has the features of both of the original declarations.

The simplest, and perhaps most common, type of declaration merging is interface merging. At the most basic level, the merge mechanically joins the members of both declarations into a single interface with the same name.

Example

```
interface Cloner {
    clone(animal: Animal): Animal;
}
interface Cloner {
```

```
    clone(animal: Sheep): Sheep;
}
interface Cloner {
    clone(animal: Dog): Dog;
    clone(animal: Cat): Cat;
}
```

The three interfaces will merge to create a single declaration as so:

```
interface Cloner {
    clone(animal: Dog): Dog;
    clone(animal: Cat): Cat;
    clone(animal: Sheep): Sheep;
    clone(animal: Animal): Animal;
}
```

> Note: Not all merges are allowed in TypeScript. Currently, classes can not merge with other classes or variables.

49) What are method overriding in TypeScript?

If subclass (child class) has the same method as declared in the parent class, it is known as method **overriding.** In other words, redefined the base class methods in the derived class or child class.

Rules for Method Overriding

- The method must have the same name as in the parent class
- The method must have the same parameter as in the parent class.
- There must be an IS-A relationship (inheritance).

Example

```
class NewPrinter extends Printer {
    doPrint(): any {
```

```
        super.doPrint();
        console.log("Called Child class.");
    }
    doInkJetPrint(): any {
        console.log("Called doInkJetPrint().");
    }
}
let printer: new () => NewPrinter;
printer.doPrint();
printer.doInkJetPrint();
```

50) What is Lambda/Arrow function?

ES6 version of TypeScript provides **shorthand** syntax for defining the anonymous function, i.e., for function expressions. These arrow functions are also called **Lambda** functions. A lambda function is a function without a name. Arrow function **omits** the function keyword.

Example

```
let sum = (a: number, b: number): number => {
        return a + b;
}
console.log(sum(20, 30)); //returns 50
```

In the above, the ?=>? is a **lambda operator** and (a + b) is the body of the function and (a: number, b: number) are **inline** parameters.

To know more click here.