Top 50 TypeScript Interview Questions

Q1. What are the Differences between TypeScript and JavaScript?

TypeScript	JavaScript
TypeScript is an Object-Oriented language	JavaScript is a Scripting language
It has a feature known as Static typing	It does not have static typing
TypeScript gives support for modules	JavaScript does not support modules
It supports optional parameter function	It does not support optional parameter function

Q2. What is TypeScript?

TypeScript is a typed superset of JavaScript that compiles to plain JavaScript. It is pure object-oriented with classes, interfaces and statically typed programming languages like C# or Java.

You will need a compiler to compile and generate the code in the JavaScript file. Basically, TypeScript is the ES6 version of JavaScript with some additional features.

Syntax:

```
var message:string = "Welcome to Edureka!"

console.log(message)
```

A 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. After the installation, the command tsc <filename>.ts compiles the TypeScript code into a plain JavaScript file.

Q3. Why do we need TypeScript?

There are different reasons why a JavaScript developer should consider using TypeScript. Some of them include:

Using new features of ECMAScript: TypeScript supports new ECMAScript standards and transpile them to ECMAScript targets of your choice. So, you can use features of ES2015 and beyond.

Static Typing: JavaScript is dynamically typed and does not know what type a variable is until it is actually instantiated at run-time. TypeScript adds type support to JavaScript.

Type Inference: TypeScript makes typing a bit easier and a lot less explicit by the usage of type inference. Even if you don't explicitly type the types, they are still there to save you from doing something which otherwise would result in a run-time error.

Better IDE Support: The development experience with TypeScript is a great improvement over JavaScript. There is a wide range of IDEs that have excellent support for TypeScript, like Visual Studio & VS code, Atom, Sublime, and IntelliJ/WebStorm.

Strict Null Checking: Errors, like cannot read property 'x' of undefined, is common in JavaScript programming. You can avoid most of these kinds of errors since one cannot use a variable that is not known to the TypeScript compiler.

Interoperability: TypeScript is closely related to JavaScript so it has great interoperability capabilities, but some extra work is required to work with JavaScript libraries in TypeScript.

Q4. Mention some of the features of TypeScript

Cross-Platform: The TypeScript compiler can be installed on any Operating System such as Windows, MacOS, and Linux.

Object-Oriented Language: TypeScript provides features like Classes, Interfaces, and Modules. Thus, it can write object-oriented code for client-side as well as server-side development.

Static Type-Checking: TypeScript uses static typing and helps type checking at compile time. Thus, you can find errors while writing the code without running the script.

Optional Static Typing: TypeScript also allows optional static typing in case you are using the dynamic typing of JavaScript.

DOM Manipulation: You can use TypeScript to manipulate the DOM for adding or removing elements.

ES 6 Features: TypeScript includes most features of planned ECMAScript 2015 (ES 6, 7) such as class, interface, Arrow functions, etc.

Q5. What are the Benefits of using TypeScript?

The Benefits of using TypeScript are:

- TypeScript is fast, simple, easy to learn and runs on any browser or JavaScript engine.
- It is similar to JavaScript and uses the same syntax and semantics.
- This helps backend developers write front-end code faster.
- You can call the TypeScript code from an existing JavaScript code. Also, it works with existing JavaScript frameworks and libraries without any issues.
- The Definition file, with .d.ts extension, provides support for existing JavaScript libraries like Jquery, D3.js, etc.
- It includes features from ES6 and ES7 that can run in ES5-level JavaScript engines like Node.js.

Q6. What are the Disadvantages of TypeScript?

TypeScript has the following disadvantages:

- TypeScript takes a long time to compile the code.
- It 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 for decades and TypeScript doesn't bring anything new.
- To use any third party library, the definition file is a must.
- Quality of type definition files is a concern.

Q7. What are the Components of TypeScript?

There are three different types of components in TypeScript which includes:

Language − It comprises of the syntax, keywords, and type annotations.

The TypeScript Compiler − This compiler (tsc) converts the instructions written in TypeScript to its JavaScript equivalent.

The TypeScript Language Service − The Language Service exposes an additional layer around the core compiler pipeline, editor-like applications. The language service supports the common set of typical editor operations.

Q8. Who developed Typescript and what is the current stable version available?

Anders Hejlsberg developed TypeScript. Also, he is 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. But, 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 the ECMAScript 2015 framework. Also, it offers support for the latest and evolving JavaScript features.

Q9. How to install TypeScript?

There are two main ways to install TypeScript tools such as:

Via npm (Node.js Package Manager) command-line tool

npm install -g typescript

By installing TypeScript via Visual Studio.

If you use Visual Studio or VS Code IDE, the easiest way to add to Visual Studio or VS Code is to search and add a package or download from the TypeScript website. Also, you can download TypeScript Tools for Visual Studio.

Q10. How do you compile TypeScript files?

The extension for any TypeScript file is .ts. And any JavaScript file is a TypeScript file as it is a superset of JavaScript. So, once you change the extension of ".js" to ".ts", your TypeScript file is ready. To compile any .ts file into .js use the following command:

tsc <TypeScript File Name>

For example, to compile "Typescriptquestions.ts"

tsc edureka.ts

And the result would be Typescriptquestions.js

Q11. Can we combine multiple .ts files into a single .js file?

Yes, we can combine multiple files. While compiling, we need to add –outFILE [OutputJSFileName] option.

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

This will compile all 3 ".ts" file and output into a single "comman.js" file.

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

If you don't provide an output file name, file2.ts and file3.ts will be compiled and the output will be placed in file1.ts. So now your file1.ts contains JavaScript code.

Q12. What are the different types of TypeScript?

The Type System represents the different types of values supported by the language. It checks the validity of the supplied values before they are stored or manipulated by the program.

It can be classified into two types such as:

Built-in: This includes number, string, boolean, void, null and undefined.

User-defined: It includes Enumerations (enums), classes, interfaces, arrays, and tuple.

Q13. List out the built-in data types in TypeScript.

In TypeScript, the built-in data types are also known as primitive data types and the list include:

Number: This represents number type values. The numbers are stored as floating-point values in TypeScript.

String: A string represents a sequence of characters stored as Unicode UTF-16 code.

Boolean: This represents a logical value. When we use the Boolean type, we get the output only in true or false.

Null: Null represents a variable whose value is undefined. It is not possible to directly reference the null type value itself.

Undefined: The Undefined type denotes all uninitialized variables.

Void: A void is the return type of the functions that do not return any type of value.

Q14. What are Variables in TypeScript and how to create them?

A variable is a named space in the memory which is used to store values. The type syntax for declaring a variable in TypeScript includes a colon (:) after the variable name, followed by its type. Similar to JavaScript, we use the var keyword to declare a variable. While declaring a variable in Typescript, certain rules must be followed-

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

Q15. What are the different ways of declaring a Variable?

There are four ways of declaring a variable:

- var [identifier] : [type-annotation] = value; //Declaring type and value in a single statement
- var [identifier] : [type-annotation]; //Declaring type without value
- var [identifier] = value; //Declaring its value without type
- var [identifier]; //Declaring without value and type

Q16. Is it possible to compile .ts automatically with real-time changes in the .ts file?

Yes, we can compile ".ts" automatically with real-time changes in the .ts file. This can be done 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 once again. Here,

we need to ensure that the command prompt must not be closed on running with –watch option.

Q17. What are the object-oriented terms supported by TypeScript?

TypeScript supports the following object-oriented terms:

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

Q18. What are Interfaces in TypeScript?

The interface is a structure that defines the contract in your application. It defines the syntax for classes to follow. It contains only the declaration of the members and it is the responsibility of the deriving class to define the members. The TypeScript compiler uses interface for type-checking and checks whether the object has a specific structure or not.

Syntax:

```
interface interface_name {

// variables' declaration

// methods' declaration

}
```

Q19. What are Classes in TypeScript? List out some of the features.

TypeScript introduced classes so that they can avail the benefits of object-oriented techniques like encapsulation and abstraction. The class in TypeScript is

compiled to plain JavaScript functions by the TypeScript compiler to work across platforms and browsers.

A class includes the following:

- Constructor
- Properties
- Methods

Example:

```
class Employee {
empID: number;
empName: string;
 constructor(ID: number, name: string) {
this.empName = name;
this.empID = ID;
}
 getSalary() : number {
return 40000;
}
}
}
```

Some of the features of a class are:

- Inheritance
- Encapsulation
- Polymorphism
- Abstraction

Q20. What are the access modifiers supported by TypeScript?

TypeScript supports access modifiers public, private and protected which determine the accessibility of a class member as given below:

Public – All the members of the class, its child classes, and the instance of the class can access.

Protected – All the members of the class and its child classes can access them. But the instance of the class can not access.

Private – Only the members of the class can access them.

If an access modifier is not specified it is implicitly public as that matches the convenient nature of JavaScript.

Q21. How is TypeScript an optionally statically typed language?

TypeScript is referred to as optionally statically typed, which means you can ask the compiler to ignore the type of a variable. Using any data type, we can assign any type of value to the variable. TypeScript will not give any error checking during compilation.

Example:

```
var unknownType: any = 4;

unknownType = "Welcome to Edureka"; //string

unknownType = false; // A boolean.
```

Q22. What are modules in TypeScript?

A module is a powerful way of creating a group of related variables, functions, classes, and interfaces, etc. It can be executed within its own scope, but not in the global scope. Basically, you cannot access the variables, functions, classes, and interfaces declared in a module outside the module directly.

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

Example:

```
module module_name{

class xyz{

export sum(x, y){

return x+y;

}

}
```

Q23. What is the difference between the internal module and the external module?

Internal Module	External Module
Internal modules 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.
Internal modules were a part of the earlier version of Typescript.	External modules are known as a module in the latest version.
These are local or exported members of other modules.	These are separately loaded bodies of code referenced using external module names.
Internal modules are declared using ModuleDeclarations that specify their name and body.	An external module is written as a separate source file that contains at least one import or export declaration.

Q24. What is namespace in Typescript and how to declare it?

Namespace groups functionalities logically. These 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.

Syntax:

```
namespace <namespace_name> {

export interface I1 { }

export class c1{ }

}
```

Q25. Does TypeScript support function overloading?

Yes, TypeScript supports function overloading. But the implementation is odd. So, when you overload in TypeScript you only have one implementation with multiple signatures.

For example:

```
class Customer {

name: string;

Id: number;

add(Id: number);

add(name:string);

add(value: any) {

if (value && typeof value == "number") {

//Do something

}

if (value && typeof value == "string") {
```

```
//Do Something
}
}
```

The first signature has one parameter of type number whereas the second signature has a parameter of type string. The third function contains the actual implementation and has a parameter of type any. The implementation then checks for the type of the supplied parameter and executes a different piece of code based on the supplier parameter type.

Q26. 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 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. To enable experimental support for decorators,you need to 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
}
}
```

Q27. What are Mixins?

In Javascript, Mixins are a way of building up classes from reusable components and then build them by combining simpler partial classes.

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. Here, function B is a mixin.

Q28. How does TypeScript support optional parameters in function?

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

Example:

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

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

In the above example, arg1 is always required, and arg2 is an optional parameter.

Q29. What is Scope variable?

The scope is a set of objects, variables, and function and the JavaScript can have a global scope variable and local scope variable.

You can declare a variable in two different scopes such as:

Local Scope Variable – It is a function object which is used within the functions

Global Scope Variable – You can use this window object out of function and within the functions

Q30. How can you debug a TypeScript file?

To debug any TypeScript file, you need a .js source map file. So, you have to 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 the last line of file1.js would be a reference of the source map file.

//# sourceMappingURL=file1.js.map

Q31. 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.Now, if you want to use some jQuery code in your .ts file:

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

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

Q32. What are the steps to include Type Definition File?

The steps involved in the process of including the Type Definition File are:

First, you have to install TSD.

$ npm install tsd -g

Next, 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 the definition file by updating the TypeScript file to point to the jQuery definition.

/// <reference path="typings/jquery/jquery.d.ts" />

$(document).ready(function() { //To Do

});

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

Q33. What is TypeScript Declare Keyword?

JavaScript libraries or frameworks don't have TypeScript declaration files. But if you want to use them in the TypeScript file without any compilation error, you have to use the declare keyword. The declare keyword is used for ambient declarations and methods where you want to define a variable that may exist elsewhere.

If you want to use the library in our TypeScript code, you can use the following code:

declare var myLibrary;

TypeScript runtime will assign the myLibrary variable as any type.

Q34. What is the Default Parameters Function in TypeScript?

Function parameters can be assigned values by default. A parameter can't be declared as optional and default both at the same time.

Example:

let discount = function (price: number, rate: number = 0.40) {

return price * rate;

}

discount(500); // Result - 200

discount(500, 0.45); // Result - 225

In the above example, rate is a default param as number in discount function. If we pass the value in the discount's rate param, it will use this otherwise use default value 0.40.

Q35. What is tsconfig.json file?

The tsconfig.json file is a file which is in JSON format. In the tsconfig.json file, you can specify different 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.

Example of 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"
```

]

}

Q36. What are Generics in TypeScript?

TypeScript Generics is a tool that provides a way of creating reusable components. It is able to create components that can work with a variety of data types rather than a single data type. Also, it 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. It uses a special kind of type variable <T> that denotes types.

Example:

```
function identity<T>(arg: T): T {
return arg;
}
let output1 = identity<string>("edureka");
let output2 = identity<number>( 117 );
console.log(output1);
console.log(output2);
```

Q37. What is the difference between interface and type statements?

Interface	Type
An interface declaration introduces a named object type	A type alias declaration introduces a name for any kind of type, including primitive, union, and intersection types

It 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
Interfaces create a new name that is used everywhere	They don’t create any new name
It can have multiple merged declarations	It cannot have multiple merged declarations

Q38. What is JSX in TypeScript?

JSX is an embeddable XML-like syntax and it is meant to be transformed into a valid JavaScript. JSX became popular with the React framework. TypeScript supports embedding, type checking, and compiling JSX directly into JavaScript.

If you want to use JSX in your file, you need to name your file with a .tsx extension and enable jsx option.

Q39. What are all the JSX modes TypeScript supports?

TypeScript consists of three JSX modes:

The preserve mode keeps the JSX as part of the output to be further consumed by another transform step. Also, the output will have a .jsx file extension. The react mode emits React.createElement, does not need to go through a JSX transformation before use, and the output will have a .js file extension.

The react-native mode is the equivalent of the preserve and it keeps all JSX, but the output has a .js file extension instead.

Q40. What are Ambients in TypeScripts and when to use them?

Ambient declarations tell the compiler about the actual source code that exists 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 files. If the source changes, the docs need to be kept updated and if the ambient declaration file is not updated, then you will get compiler errors. Also, it allows us to safely and easily use existing popular JavaScript libraries like jquery, angularjs, nodejs, etc.

Q41. What is a TypeScript Map file?

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

Q42. 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 in other languages 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 = 007;

let employeeCode = <number> code;

console.log(typeof(employeeCode)); //Output: number
```

Q43. What are 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.

Q44. What are the rules to declare Rest parameters? Give an example.

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.

Example:

```
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(2, 4);
let result2 = sum(2,4,6,8);
```


Q45. 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 conflicted with JSX.

Example:

```
let empCode: any = 007;
let employeeCode = code as number;
```

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

Q46. Explain Enum in TypeScript.

Enums or enumerations are a TypeScipt data type that allows us to define a set of named constants. Using enums 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.Male); // Output: 0
//We can also access an enum value by it's number value.
console.log(Gender[1]); // Output: Female
```


Q47. Explain Relative and Non-relative module imports.

Relative	Non-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:** import * as $ from "jquery";	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 Entry from "./components/Entry";

import { Component } from "@angular/core";	import {DefaultHeaders} from "../constants/http";

Q48. What is an anonymous function?

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

Example:

```
let myAdd = function(x: number, y: number): number {

return a+b;

};

console.log(myAdd())
```

Q49. What is method overriding in TypeScript?

If the subclass or the child class has the same method as declared in the parent class, it is known as method overriding. Basically, it redefines 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
- It must have the same parameter as in the parent class.
- There must be an IS-A relationship or inheritance.

Q50. 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. Whereas, the arrow function omits the function keyword.

Example:

```
let sum = (x: number, y: number): number => {
return x + y;
}
console.log(sum(10, 20)); //returns 30
```

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