TypeScript Interview Questions:

Q1: What are the key advantages of using TypeScript over JavaScript?

APPROACH:

Discuss TypeScript's features that enhance code reliability and maintainability.

ANSWER:

TypeScript introduces static typing, which allows for catching errors at compile time rather than at runtime, reducing runtime errors significantly. It also includes features such as interfaces and generics, enhancing code quality and reusability. TypeScript's tooling support with advanced autocompletion, navigation, and refactoring features makes it a robust choice for large-scale projects.

Q2: How does TypeScript support object-oriented programming?

APPROACH:

Explain TypeScript's object-oriented features and compare them with traditional JavaScript.

ANSWER:

TypeScript fully supports object-oriented programming concepts including classes, interfaces, inheritance, and access modifiers such as public, private, and protected. This structured approach is less prone to errors and more manageable compared to JavaScript's prototypal inheritance.

Q3: Can you describe the TypeScript compilation process?

APPROACH:

Outline how TypeScript code is transpiled into JavaScript.

ANSWER:

TypeScript code is transpiled into JavaScript using the TypeScript compiler or via build tools like Webpack or Babel. During this process, the compiler checks for type errors and emits JavaScript that is compatible with older browsers depending on the target settings in the tsconfig.json file.

Q4: How does TypeScript handle type inference?

APPROACH:

Explain the concept of type inference in TypeScript and its benefits.

ANSWER:

TypeScript is capable of inferring the types of variables when they are initialized, which simplifies the code without compromising the safety and benefits of static typing. This feature reduces the need to explicitly define a type every time, thereby making the code cleaner and easier to read.

Q5: What are union and intersection types in TypeScript?

APPROACH:

Provide an explanation and examples for union and intersection types.

ANSWER:

Union types allow a variable to be one of several types (e.g., `number | string`), enabling more flexible code structures. Intersection types combine multiple types into one (e.g., `TypeA & TypeB`), which is particularly useful when mixing in multiple sources of types or when enhancing types with additional functionality.

Q6: How does TypeScript handle `null` and `undefined`?

APPROACH:

Discuss TypeScript's type-checking behavior for `null` and `undefined`, and explain how it can be configured via TypeScript's compiler options.

ANSWER:

TypeScript treats `null` and `undefined` as distinct types. By default, `null` and `undefined` can be assigned to any type, making all types effectively nullable. However, TypeScript can be configured to handle `null` and `undefined` more strictly by enabling the `strictNullChecks` option in the `tsconfig.json` file. When enabled, `null` and `undefined` are only assignable to `any` and their respective types, enhancing type safety by preventing many common errors. For instance, a function expecting a string cannot be passed `null` or `undefined` unless the type is explicitly defined as `string | null | undefined`. This setting helps in writing more predictable code and aids in the prevention of runtime errors related to null or undefined access.

Q7: Can TypeScript be used for backend development?

APPROACH:

Discuss the feasibility and benefits of using TypeScript for backend applications, especially in Node.js environments.

ANSWER:

Yes, TypeScript can be effectively used for backend development, primarily through Node.js, a popular JavaScript runtime. TypeScript compiles down to JavaScript, which means it can run anywhere JavaScript runs, including on the server side. Using TypeScript in backend development offers several advantages such as improved code reliability through strong typing, enhanced developer productivity with better tooling support, and easier maintenance due to clear code structure and object-oriented features. Many Node.js frameworks and libraries, like Express, NestJS, and TypeORM, provide TypeScript support out of the box, facilitating its adoption and integration into the backend. Additionally, TypeScript's compatibility with modern JavaScript features and its strong typing system make it ideal for large-scale and complex server-side applications.

Q8: How do you create and use enums in TypeScript?

APPROACH:

Describe the purpose of enums in TypeScript and provide an example of their usage.

ANSWER:

Enums in TypeScript are constructs that provide a way to define sets of named constants. Using enums makes it easier to document intent or create a set of distinct cases. They encourage clearer and more maintainable code by allowing developers to use descriptive names for sets of numeric or string values. Enums help to ensure that the code remains readable and less error-prone by minimizing the use of arbitrary constants or unexplained literal values throughout the application. For instance, in a web application handling user permissions, you might define an enum for user roles like this:

```
enum UserRole {
    Guest,
    User,
    Moderator,
    Admin
}
```

This enum can be used to set roles in user objects or to check permissions in a straightforward way, reducing errors that may arise from using literal strings or numbers:

```
function checkAccess(userRole: UserRole): string {
    if (userRole === UserRole.Admin) {
        return "Access granted for all operations.";
    } else if (userRole >= UserRole.Moderator) {
        return "Access granted for moderation.";
    } else {
        return "Read-only access.";
    }
}

let userRole: UserRole = UserRole.Moderator;
console.log(checkAccess(userRole)); // Outputs: "Access granted for moderation."
```

In this example, `UserRole` helps to enforce type safety in function parameters and increases the code's self-documentation properties.