## Beginner-Level TypeScript Questions:

1. **What is TypeScript, and how is it different from JavaScript?**
   - Explanation of TypeScript as a statically-typed superset of JavaScript that adds optional types, interfaces, and other features for improving code quality.
2. **How do you declare types in TypeScript?**
   - Syntax for declaring basic types such as `string`, `number`, `boolean`, `any`, `void`, `null`, `undefined`, and custom types using interfaces and type aliases.
3. **What is the `any` type in TypeScript, and when should it be used?**
   - Explanation of the `any` type, its purpose for bypassing static type checking, and why overuse can negate the benefits of TypeScript.
4. **What is the difference between `interface` and `type` in TypeScript?**
   - How interfaces are used to define the shape of objects, classes, and functions, while `type` can define unions, intersections, primitives, and complex types.
5. **What are TypeScript Enums, and how do you define them?**
   - Explanation of enums in TypeScript, the difference between numeric and string enums, and examples of how to use them to define a set of named constants.
6. **How do you define optional properties in an interface or type?**
   - Using the `?` operator to define properties that are not required in TypeScript interfaces and type aliases.
7. **What are the benefits of using TypeScript over plain JavaScript?**
   - Discussion of the benefits such as type safety, better tooling (intellisense), early error detection, and improved maintainability in large codebases.
8. **What is the `never` type in TypeScript?**
   - Explanation of the `never` type, when a function never returns (e.g., in cases of exceptions or infinite loops).
9. **What is TypeScript's `readonly` modifier, and how is it used?**
   - How to use the `readonly` modifier to create properties that cannot be reassigned after the object has been created.
10. **How does TypeScript's type inference work?**
    - Explanation of how TypeScript can automatically infer the type of variables and return types of functions without explicit type annotations.
11. **What are union types in TypeScript?**
    - Defining union types using the `|` operator to allow a variable to hold multiple possible types.
12. **What are type aliases in TypeScript?**
    - Using the `type` keyword to create type aliases for more complex or repetitive type structures.
13. **How does TypeScript handle null and undefined types?**
    - Explanation of how `null` and `undefined` can be explicitly handled in TypeScript and how to enable strict null checks with `strictNullChecks`.

14. **What is TypeScript's `unknown` type, and how is it different from `any`?**
    - Discussion of the `unknown` type, which is safer than `any` because values need to be narrowed down before performing operations on them.
15. **What is a tuple in TypeScript, and how is it different from an array?**
    - Explanation of tuples as fixed-length arrays with predefined types for each element, compared to regular arrays where the type can be uniform or flexible.
16. **What is type assertion in TypeScript, and when should it be used?**
    - Syntax for type assertion (`as` or `<type>`) and when it is useful for telling TypeScript to treat a value as a specific type.
17. **What is the purpose of the `void` type in TypeScript?**
    - Explanation of `void`, typically used as a return type for functions that do not return a value.
18. **What is structural typing in TypeScript?**
    - Discussion of TypeScript's structural type system, where compatibility is based on the shape of the data rather than explicit declarations.
19. **How does TypeScript handle function overloading?**
    - Explanation of function overloads, how to declare multiple function signatures, and how TypeScript resolves the correct implementation.
20. **How do you declare and work with generics in TypeScript?**
    - Introduction to generics, how to define reusable functions and classes with generic types, and examples of using generics with constraints.

---

## Intermediate-Level TypeScript Questions:

21. **What are generics in TypeScript, and how do they improve code reusability?**
    - Explanation of generic types and how they allow functions, classes, or interfaces to work with any data type while preserving type safety.
22. **What are mapped types in TypeScript?**
    - Explanation of how mapped types allow you to create new types by transforming existing ones (e.g., making all properties optional or readonly).
23. **What is `keyof` in TypeScript, and how can it be used?**
    - Explanation of the `keyof` operator, which retrieves the keys of an object type as a union of string literal types.
24. **What are utility types in TypeScript?**
    - Overview of built-in utility types like `Partial<T>`, `Required<T>`, `Readonly<T>`, `Pick<T, K>`, `Record<K, T>`, and `Omit<T, K>` and how they simplify common type transformations.
25. **How do TypeScript's `extends` and `implements` keywords differ in class inheritance?**
    - Explanation of how `extends` is used for class inheritance and `implements` is used to enforce contracts via interfaces.

26. **What is TypeScript's `this` type, and how is it useful?**
    - Explanation of how TypeScript's `this` type can be used in methods to refer to the current class instance, and how to use it for better type safety in fluent APIs.
27. **What are discriminated unions in TypeScript?**
    - Explanation of discriminated (tagged) unions, using a common literal property to differentiate between multiple possible object shapes in a type-safe way.
28. **How does TypeScript handle intersection types?**
    - Explanation of intersection types using the `&` operator to combine multiple types into a single type that includes all their properties.
29. **What is TypeScript's module resolution process?**
    - Explanation of how TypeScript resolves modules in a project, including its support for CommonJS, AMD, ES modules, and how configuration options like `paths` and `baseUrl` affect module resolution.
30. **How can you improve the build performance of large TypeScript projects?**
    - Techniques such as using `tsconfig` options like `skipLibCheck`, incremental builds, project references, and leveraging tools like `tsc` in watch mode.
31. **How do you work with decorators in TypeScript, and what are their use cases?**
    - Explanation of decorators in TypeScript, their syntax, and their use cases in class methods, properties, or parameter decoration (e.g., in Angular).
32. **What is declaration merging in TypeScript?**
    - Discussion of how TypeScript allows merging of multiple declarations (e.g., interfaces, functions) with the same name, useful for augmenting types in third-party libraries.
33. **What are conditional types in TypeScript?**
    - Explanation of conditional types using the syntax `T extends U ? X : Y` and how they enable flexible type transformations based on conditions.
34. **How does TypeScript handle asynchronous programming with `async` and `await`?**
    - Overview of TypeScript's support for async/await, how it relates to Promises, and how to type asynchronous functions.
35. **What is `Partial<T>` in TypeScript, and when should you use it?**
    - Explanation of the `Partial<T>` utility type, which makes all properties of an interface optional, and its use cases in update operations or optional configuration objects.
36. **What are type guards in TypeScript?**
    - Explanation of type guards (`typeof`, `instanceof`, user-defined type predicates) and how they help in narrowing down types within control flow.
37. **What is TypeScript's `infer` keyword, and how is it used?**
    - Explanation of the `infer` keyword, used in conditional types to infer types dynamically during type resolution.
38. **What is the difference between `public`, `private`, and `protected` access modifiers in TypeScript?**

    - Explanation of access modifiers in TypeScript and how they control the visibility and accessibility of class properties and methods.
39. **How do you ensure type safety when using third-party libraries that don't provide TypeScript types?**
    - Explanation of using `@types` packages, `declare` keyword, or writing custom type definitions for external libraries.
40. **What are module augmentation and ambient declarations in TypeScript?**
    - Explanation of how module augmentation allows adding properties or methods to existing types and how ambient declarations (`declare module`) are used to extend global modules.

---

## Advanced-Level TypeScript Questions:

41. **How do you enforce immutability in TypeScript?**
    - Discussion of using `readonly` and `ReadonlyArray<T>` or third-party libraries like `Immutable.js` to create immutable data structures.
42. **How do you implement advanced type-safe patterns such as builders or factories in TypeScript?**
    - Techniques for building complex type-safe patterns using generics, mapped types, and utility types in TypeScript.
43. **What are template literal types in TypeScript, and how are they useful?**
    - Explanation of how template literal types (introduced in TypeScript 4.1

44. **What are conditional types and how do you create them?**
- Explanation of conditional types, their syntax, and examples of how they are used to create types that depend on other types.
45. **How do you create a type-safe API in TypeScript?**
- Discussion of how to define interfaces for request and response objects, use generics for flexibility, and implement strong type checking for API interactions.
46. **What are `never` and `void` types, and when would you use them?**
- Explanation of the differences between `never` (used for functions that throw errors or never complete) and `void` (indicating no return value) types.
47. **How can you create a recursive type in TypeScript?**
- Explanation of how to define types that refer to themselves, such as tree structures, and examples of recursive type definitions.
48. **What are the differences between `Function` and `(...) => void` types in TypeScript?**
- Discussion of how to declare functions with the `Function` type versus using a specific function signature, and why it's generally better to use specific signatures.
49. **How does TypeScript's type system differ from other languages like Java or C#?**

- Comparison of TypeScript's structural typing versus nominal typing in languages like Java or C#, and implications for type compatibility and safety.

50. **What is the `infer` keyword and how do you use it in conditional types?**
- Explanation of how to use `infer` to create types that infer values based on the conditions provided in a conditional type.

51. **What is the purpose of `this` in TypeScript and how does it differ from JavaScript?**
- Discussion of the `this` type in TypeScript and how it provides better type safety for methods in classes and functions.

52. **How do you create a union type with a specific structure?**
- Explanation of how to define a union type consisting of different object shapes, and examples illustrating the concept.

53. **What are utility types in TypeScript and how can they simplify your code?**
- Overview of commonly used utility types (`Partial`, `Required`, `Readonly`, `Pick`, `Record`, `Omit`) and how they can simplify type transformations.

54. **How can you use TypeScript with React (or another framework)?**
- Discussion on how to integrate TypeScript into a React project, typing props, state, and context, and using TypeScript's features for component development.

55. **What is a type predicate, and how can it be used in a function?**
- Explanation of type predicates, their syntax, and how to use them to create functions that narrow down types based on specific conditions.

56. **How do you implement polymorphism in TypeScript?**
- Discussion of how to achieve polymorphism through interfaces, base classes, and method overriding in TypeScript.

57. **What is the difference between a static type and a dynamic type in TypeScript?**
- Explanation of static types (checked at compile time) versus dynamic types (checked at runtime) and their implications for type safety.

58. **How can you create custom types in TypeScript using `mapped types`?**
- Explanation of how to create new types based on existing types using mapped types to transform property types or keys.

59. **What are the implications of using `strictNullChecks` in TypeScript?**
- Discussion on the benefits and challenges of enabling `strictNullChecks`, including how it affects handling `null` and `undefined` values.

60. **How do you leverage namespaces in TypeScript?**
- Explanation of how to create and use namespaces to organize code, avoid naming conflicts, and encapsulate functionality.

61. **How can you define global types or augment existing types in TypeScript?**
- Discussion of how to declare global types using ambient declarations and how to extend existing types with additional properties or methods.

62. **What are the advantages of using TypeScript in large-scale applications?**
- Discussion of how TypeScript improves maintainability, scalability, and developer productivity in large codebases through static type checking and enhanced tooling.

63. **What is type inference and how does it work in TypeScript?**

- Explanation of how TypeScript automatically infers types based on context and examples of situations where type inference occurs.

64. **How can you enforce function arguments to be a specific type using generics?**

- Explanation of how to create generic functions that enforce specific types for arguments and return types.

65. **How does TypeScript handle module loading and imports?**

- Overview of how TypeScript supports different module systems (CommonJS, ES Modules) and how to configure module resolution in `tsconfig.json`.

66. **What is an ambient declaration in TypeScript?**

- Explanation of ambient declarations (`declare`) and their role in providing type information for global variables or external libraries.

67. **What are decorators, and how do you implement them in TypeScript?**

- Discussion of the decorator syntax in TypeScript, use cases for decorators (e.g., logging, validation), and how they modify class behavior.

68. **How do you handle type compatibility and structural typing in TypeScript?**

- Explanation of how TypeScript uses structural typing for type compatibility and examples demonstrating the concept.

69. **What is the role of `@ts-ignore` and `@ts-expect-error` in TypeScript?**

- Discussion of how to use these comments to suppress TypeScript errors and the differences between them.

70. **How do you manage and install type definitions for third-party libraries?**

- Overview of using DefinitelyTyped and the `@types` scope for managing type definitions for popular JavaScript libraries.

## Behavioral and Situational Questions:

71. **Can you describe a challenging problem you solved using TypeScript?**

- A chance to discuss a real-world application of TypeScript to address a complex issue.

72. **How do you stay updated with the latest TypeScript features and best practices?**

- Discussion on resources, communities, or practices for staying current with TypeScript developments.

73. **How would you refactor a large JavaScript codebase to TypeScript?**

- Overview of the steps and considerations involved in migrating an existing JavaScript project to TypeScript.

74. **Can you describe your experience with TypeScript in a team environment?**

- Insights into collaborative practices, code reviews, and how to handle TypeScript-related challenges within a team.

75. **What strategies do you use to ensure code quality and consistency in TypeScript projects?**

- Discussion of best practices, tools (e.g., linters, formatters), and methodologies to maintain high-quality TypeScript code.
- **What is type assertion in TypeScript, and how is it different from type casting?**

- Explanation of type assertions (`as` syntax or angle bracket syntax) and how it differs from type casting in other programming languages.

77. **How can you create a type-safe way to handle dynamic keys in an object?**

- Discussion on using index signatures and mapped types to define types for objects with dynamic keys.

78. **What are `template literal types`, and how do you use them?**

- Explanation of template literal types and how to create string literal types that concatenate other string literal types.

79. **How can you implement function overloading in TypeScript?**

- Explanation of how to define multiple signatures for a single function and how TypeScript resolves them.

80. **What are some common design patterns you can implement using TypeScript?**

- Discussion of patterns like Singleton, Factory, and Observer and how TypeScript enhances their implementation with type safety.

81. **How do you create a type-safe version of the `Array.prototype.map` function?**

- Explanation of how to implement a type-safe mapping function that maintains the types of the input and output arrays.

82. **What are `enum` types in TypeScript, and when would you use them?**

- Discussion of how to define enums in TypeScript, their advantages, and scenarios where they are particularly useful.

83. **How can you utilize `async/await` in TypeScript, and what are the type implications?**

- Explanation of how to use `async/await` for handling asynchronous operations and the type annotations that may be needed.

84. **How do you handle errors in asynchronous code using TypeScript?**

- Discussion of best practices for error handling with `try/catch` and how to define types for potential errors.

85. **What is the purpose of `declare module` and when would you use it?**

- Explanation of how to declare modules in TypeScript for third-party libraries that do not have type definitions.

86. **How can you implement a type-safe Redux store in TypeScript?**

- Discussion of how to type the state, actions, and reducers in a Redux store to ensure type safety across the application.

87. **What are intersection types, and how do you use them?**

- Explanation of how intersection types combine multiple types into one and examples of their use cases.

88. **How do you create a `Record` type and when would you use it?**

- Discussion of how to create a `Record` type in TypeScript and scenarios where it simplifies type definitions.

89. **What is the difference between `any`, `unknown`, and `never` types?**

- Detailed explanation of the three types, their use cases, and why you should prefer `unknown` over `any`.

90. **How can you leverage `const` assertions in TypeScript?**
- Discussion on how to use `const` assertions to create immutable objects or arrays and their impact on type inference.

91. **What is a `Mapped Type` in TypeScript, and how can you use it for transformation?**
- Explanation of how to create mapped types to transform existing types by modifying properties.

92. **How can you handle deep object types in TypeScript?**
- Discussion of techniques for defining and manipulating deeply nested object types while maintaining type safety.

93. **What are the best practices for managing TypeScript configurations in large projects?**
- Overview of strategies for organizing `tsconfig.json` files, handling multiple configurations, and using project references.

94. **How do you create type-safe API responses using TypeScript?**
- Explanation of defining types for API responses and how to enforce structure and type safety when consuming APIs.

95. **What are the limitations of TypeScript, and how do you work around them?**
- Discussion of common limitations or pitfalls in TypeScript and strategies to mitigate them.

96. **How do you enforce immutability in TypeScript types?**
- Overview of strategies for defining and enforcing immutable types, using `Readonly` utility types, and structuring your types.

97. **What are `namespace` and `module` keywords in TypeScript, and how do they differ?**
- Explanation of the purpose and differences between `namespace` and `module` in organizing TypeScript code.

98. **How do you create a generic type that represents a tuple of fixed lengths?**
- Discussion on how to define generic types that ensure tuples have specific lengths and types.

99. **What are the benefits of using TypeScript with GraphQL?**
- Overview of how TypeScript enhances type safety in GraphQL queries, mutations, and schema definitions.

100. **How do you manage dependencies between types in TypeScript?** - Explanation of how to handle circular dependencies and ensure proper type resolution in complex projects.

## Behavioral and Situational Questions (Continued):

101. **Describe a time you used TypeScript to improve a project's maintainability.** - A chance to share an example of using TypeScript's features to enhance code quality and maintainability.
102. **How do you approach debugging TypeScript code? -** Discussion on strategies, tools, and best practices for debugging TypeScript applications effectively.

103. **What challenges have you faced when migrating a JavaScript project to TypeScript? -** Insights into migration experiences, obstacles encountered, and how they were overcome.
104. **How do you handle versioning of type definitions in a TypeScript project? -** Overview of best practices for managing and versioning type definitions, especially in larger teams or libraries.
105. **What resources or strategies do you recommend for learning advanced TypeScript? -** Discussion of books, online courses, or community resources that provide in-depth knowledge of TypeScript.