98 TypeScript interview questions to hire top developers

Questions

1. What is TypeScript, simply put?

2. Why might someone choose TypeScript over JavaScript?

3. Can you describe a basic TypeScript type?

4. How do you declare a variable with a specific type in TypeScript?

5. What's the deal with 'any' in TypeScript? When would you use it?

6. Explain what a TypeScript interface is.

7. How are interfaces useful in TypeScript development?

8. What are TypeScript enums and why are they helpful?

9. What's the difference between `null` and `undefined` in TypeScript?

10. Tell me about TypeScript's `void` type.

11. What is a TypeScript class, and how does it relate to JavaScript classes?

12. How do you compile a TypeScript file into JavaScript?

13. What are generics in TypeScript, in simple terms?

14. Why would you use generics in TypeScript?

15. What is type inference in TypeScript?

16. How does TypeScript help catch errors during development?

17. What's the purpose of a `tsconfig.json` file?

18. Can you explain what a union type is in TypeScript?

19. What are literal types in TypeScript?

20. How do you define a function's parameter types and return type in TypeScript?

21. What is the difference between type assertion and type casting in Typescript?

22. Explain what is meant by 'duck typing' in the context of TypeScript?

23. What are mapped types in TypeScript and how are they useful?

24. Describe how you would use TypeScript with React?

25. What are decorators in TypeScript and what problem do they solve?

26. How does the `never` type function in TypeScript?

27. What are conditional types in Typescript? Explain with an example.

28. How do you prevent a variable from being reassigned in TypeScript?

29. Describe the difference between `readonly` and `const` in Typescript.

30. What is a tuple in TypeScript, and when would you use one?

31. How do you define and use conditional types in TypeScript? Can you provide a practical example?

32. Explain the difference between mapped types and template literal types. When would you use one over the other?

33. What are discriminated unions and how do they improve type safety? Show with code.

34. Describe how to use the `infer` keyword in conditional types. What problem does it solve?

35. How does TypeScript handle variance (covariance, contravariance, invariance) in function parameters? Give examples.

36. Explain the concept of 'declaration merging' in TypeScript. Provide a use case where it is beneficial.

37. What are ambient declarations (`.d.ts` files) used for? How do you write one for a JavaScript library?

38. How do you create and use custom type guards? Why are they important?

39. Explain how to use generics with interfaces and classes in TypeScript. Provide example of a generic repository.

40. What is the purpose of the `keyof` operator in TypeScript? How can it be used with mapped types?

41. Describe how to use the `Partial`, `Readonly`, `Required`, and `Pick` utility types. Give use cases.

42. How can you prevent TypeScript from implicitly assigning the `any` type? What are the best practices?

43. Explain how to work with tuples in TypeScript, including labeled tuples and rest elements in tuples.

44. How do you define and use recursive types in TypeScript? Provide an example, like a tree structure.

45. What are declaration files and how are they used in TypeScript projects that include JavaScript libraries without TypeScript definitions?

46. Explain the concept of 'type widening' and 'type narrowing' in TypeScript. Give examples of how they occur.

47. Describe how to use the `NonNullable` utility type. In what scenarios is it most useful?

48. How can you extend existing interfaces or types in TypeScript to add new properties or methods? Provide examples of interface inheritance.

49. Explain how to use index signatures in TypeScript. What problem do they solve when dealing with dynamic data?

50. How do you define and use tagged union types effectively? Illustrate with an example like handling different API response types.

51. Explain the use of `unknown` type over `any`. What are the benefits of using `unknown`?

52. How do you handle function overloading in TypeScript? Provide a practical use case.

53. Explain conditional types in TypeScript and provide a use case where they are particularly helpful. Can you show a practical example?

54. What are mapped types in TypeScript? Demonstrate how you can use them to create new types based on existing ones, modifying properties as you go.

55. How does TypeScript's `infer` keyword work within conditional types? Give an example of how you'd use it to extract a type from a function's return type.

56. Describe the use of declaration merging in TypeScript. When would you use it, and what are the potential pitfalls?

57. Explain the concept of discriminated unions in TypeScript. How do they improve type safety when dealing with different object shapes?

58. What are recursive types in TypeScript? Illustrate a scenario where you'd need to define a recursive type, like representing a nested object structure.

59. How do you use type guards in TypeScript, and why are they important for narrowing down types within conditional blocks?

60. What is the difference between `Pick`, `Omit`, `Partial` and `Required` utility types in TypeScript? Provide examples of when you might use each.

61. Explain how you can create a custom utility type in TypeScript. What are some common use cases for custom utility types?

62. Describe the use of namespaces and modules in TypeScript. When would you use one over the other, especially in larger projects?

63. How does TypeScript handle generics? Explain the concept of generic constraints and provide an example.

64. Explain the concept of 'declaration files' (`.d.ts`) in TypeScript. Why are they important, and how do they work with JavaScript libraries?

65. How do you use decorators in TypeScript? Provide an example of creating and using a class decorator.

66. What are the different ways to handle `null` and `undefined` in TypeScript to prevent errors? Explain the use of strict null checking.

67. Explain what is meant by variance in TypeScript (covariance, contravariance, invariance). How can this impact type compatibility?

68. How does TypeScript's `Readonly` type modifier work, and how does it differ from `const`? Give example use cases.

69. Explain the concept of tagged types (branded types) in TypeScript and how they can be used for enhanced type safety.

70. Describe how you would implement a type-safe event emitter in TypeScript using generics and discriminated unions.

71. Explain how you can use TypeScript to create a type-safe wrapper around a JavaScript library that doesn't have its own type definitions.

72. Describe the challenges and solutions when migrating a large JavaScript codebase to TypeScript.

73. Explain the concept of 'declaration merging' in TypeScript and provide a practical use case where it simplifies development.

74. How does TypeScript's conditional types feature enable advanced type-level programming? Give an example.

75. Describe the differences between TypeScript's 'namespace' and ES modules. When would you use one over the other?

76. What are discriminated unions in TypeScript, and how do they improve type safety when dealing with complex data structures?

77. How can you leverage TypeScript's 'readonly' modifier to enforce immutability at compile time?

78. Explain how to create and use custom type guards in TypeScript to narrow down the type of a variable within a specific scope.

79. Describe the concept of 'covariance' and 'contravariance' in TypeScript, and how they relate to type compatibility.

80. How do you handle errors in asynchronous TypeScript code, ensuring proper type safety and avoiding potential runtime exceptions?

81. What is the purpose of the 'unknown' type in TypeScript, and when should you use it instead of 'any'?

82. Explain how to use mapped types in TypeScript to transform existing types into new ones based on specific transformations.

83. Describe the differences between 'interface' and 'type' aliases in TypeScript, highlighting their respective strengths and weaknesses.

84. How can you use TypeScript's 'decorators' feature to add metadata or modify the behavior of classes, methods, or properties?

85. Explain how to create and use generic constraints in TypeScript to restrict the types that can be used with a generic type parameter.

86. Describe the concept of 'definite assignment assertion' in TypeScript, and when it is appropriate to use it.

87. How can you use TypeScript's 'module augmentation' feature to add new properties or methods to existing modules or libraries?

88. Explain how to create and use custom JSX typings in TypeScript to ensure type safety when working with React or other JSX-based frameworks.

89. Describe the purpose of the 'declare' keyword in TypeScript, and how it is used to interact with existing JavaScript code or external libraries.

90. How can you use TypeScript's 'project references' feature to improve build times and code organization in large projects?

91. Explain the concept of 'liveness' and 'reachability' in garbage collection, and how TypeScript can help prevent memory leaks.

92. Describe the differences between structural typing and nominal typing, and how TypeScript's structural typing system affects code compatibility.

93. How do you use TypeScript to define types for complex data structures, such as trees or graphs, ensuring type safety and preventing runtime errors?

94. Explain how you would design and implement a type-safe event emitter using TypeScript's advanced type system features.

95. Describe the challenges of migrating a large JavaScript codebase to TypeScript, and the strategies you would use to minimize disruption and ensure a smooth transition.

96. How do you use TypeScript to create a domain-specific language (DSL) with custom syntax and semantics, providing a more expressive and type-safe way to solve specific problems?

97. Explain how TypeScript's type system can be used to enforce security policies and prevent common vulnerabilities, such as cross-site scripting (XSS) or SQL injection.

98. Describe how you would use TypeScript to build a type-safe API client for a RESTful web service, automatically generating types from the API schema.