TypeScript Practice Questions (Basic to Advanced)

? BASIC LEVEL (6?15)

6. Create a number[] array of your 5 favorite numbers.
7. Create a function add that takes two numbers and returns their sum.
8. Write a function greet(name?: string) that prints "Hello, Guest" if no name is provided.
9. Use type to define a custom type User with id: number, name: string, and email: string.
10. Use interface to define a Product with id, name, price.
11. Create a union type Status = "loading" | "success" | "error" and assign it to a variable.
12. Create a literal type for days of the week ("Mon" | "Tue" | ...).
13. Create a function logMessage(msg: string | number) that handles both types.
14. Declare a tuple type to represent a [string, number] pair for a student name and score.
15. Create an enum Direction with "Up", "Down", "Left", "Right".

? INTERMEDIATE LEVEL (16?30)

16. Write a Car interface with brand, year, and an optional model.
17. Create a readonly property inside an object using interface.
18. Write a class Person with name, age, and a greet() method.
19. Make a BankAccount class with a private balance and methods to deposit() and getBalance().
20. Extend a class Animal into Dog and Cat with their own methods.
21. Add a getter and setter to a class User to validate and format the email.
22. Create a custom type guard function: isString(x: unknown): x is string.
23. Use in keyword to differentiate between Admin and User types.
24. Use instanceof to check class instances at runtime.
25. Write a generic function identity<T>(value: T): T.
26. Create a generic type Box<T> with a content: T.
27. Use keyof to create a function that accepts only keys from an object.
28. Use typeof to create a type from an existing object.
29. Use ReturnType<typeof fn> to infer a function's return type.
30. Use Partial<T> to make all properties of a type optional.

? ADVANCED LEVEL (31?45)

31. Implement a custom Readonly<T> utility type using mapped types.

32. Create a mapped type Nullable<T> that makes all fields T | null.

33. Build a Pick<T, K> clone using mapped types and keyof.

34. Create a type Without<T, K> that removes keys K from type T.

35. Create a conditional type IsNumber<T> that returns "yes" if T is number, else "no".

36. Use a generic function to filter an array of items by a key.

37. Create a function pluck<T, K extends keyof T>(obj: T, key: K): T[K].

38. Build an abstract class Shape with abstract getArea(), then extend it with Rectangle and Circle.

39. Implement a discriminated union for handling different form inputs (type: "text" | "checkbox" | "date").

40. Create a function that accepts an object with any shape and logs all keys and their types.

41. Create a type-safe configuration object using as const.

42. Use Record<K, T> to create a type for a lookup object of id: string to User.

43. Create a function that narrows a value using typeof, in, and instanceof together.

44. Implement a class Logger that logs different data types using generics.

45. Create a strongly-typed React component (if you're using React) with Props and typed useState.