Intermediate & Advanced TypeScript Questions with Answers

A curated list of intermediate to advanced TypeScript interview questions for experienced Angular and frontend developers.

■ What is the difference between 'unknown' and 'any' in TypeScript?

■ ‘any’ disables type checking entirely, allowing any operation. ‘unknown’ is safer — you must perform type checking before using the value. It forces explicit type guards, reducing runtime errors.

■ What are utility types in TypeScript and name a few commonly used ones?

■ Utility types are built-in helpers that transform existing types. Common ones include Partial, Required, Readonly, Pick, Omit, and Record. They help in creating flexible, reusable interfaces.

■ How does structural typing differ from nominal typing?

■ TypeScript uses structural typing, meaning compatibility is based on shape (the structure of the type), not the name. Two types with the same properties are considered compatible even if defined separately.

■ What is the difference between interface and type alias?

■ Both define object shapes, but interfaces can be extended (merged) multiple times, while type aliases cannot. However, types are more flexible — they can represent unions, intersections, and primitives.

■ Explain what mapped types are and when to use them.

■ Mapped types allow you to create new types by transforming properties of an existing type. For example, making all properties optional or readonly. They’re used in scenarios like form models or API response transformations.

■ What are conditional types in TypeScript?

■ Conditional types allow type decisions at compile time: `T extends U ? X : Y`. They are useful for defining dynamic relationships, e.g., returning element type if input is an array.

■ What is the infer keyword used for?

■ ‘infer’ is used inside conditional types to extract and infer a type. For example, you can infer the return type of a function or the type of elements inside an array.

■ What are discriminated unions and how are they useful?

■ They combine multiple types into one with a common discriminant property (like kind or type). TypeScript uses this field for narrowing, improving type safety in switch or conditional statements.

■ Explain how decorators work in TypeScript.

■ Decorators are functions that can modify classes, methods, or properties. They're executed at design time and widely used in Angular for dependency injection and metadata annotation.

■ What is the difference between abstract classes and interfaces?

■ Interfaces define contracts, while abstract classes provide partial implementation. Abstract classes can include logic, constructors, and access modifiers; interfaces cannot.

■ How does TypeScript support asynchronous code typing?

■ TypeScript supports async functions via the `Promise` type. You can define function signatures that return `Promise`, ensuring correct type inference for awaited results.

■ What is declaration merging in TypeScript?

■ When multiple declarations with the same name are encountered, TypeScript merges them into a single definition. Commonly used for extending interfaces or adding new properties to modules.

■ What are ambient declarations in TypeScript?

■ Ambient declarations (`declare`) tell TypeScript that a variable, function, or module exists elsewhere (e.g., from an external script). They are used in definition files (*.d.ts).

■ Explain keyof and typeof operators in TypeScript.

■ `keyof` returns a union of property names of a type. `typeof` gives the type of a variable or object at compile time. Together, they help create type-safe accessors or mappers.

■ How does TypeScript handle module resolution?

■ Module resolution determines how imports are located. It supports two strategies: Node and Classic. Node mimics Node.js resolution logic, while Classic is used for older codebases.

■ What are type guards?

■ Type guards are runtime checks that refine a variable's type within a scope. They use `typeof`, `instanceof`, or custom predicates like `value is MyType`.

■ What is the purpose of 'never' type in TypeScript?

■ The 'never' type represents values that never occur, such as functions that throw errors or infinite loops. It ensures all code paths are handled exhaustively.

■ What are intersection types in TypeScript?

■ Intersection types combine multiple types into one. A variable of intersection type must satisfy all combined type constraints, useful for mixins or composing features.

■ Explain the difference between optional chaining and nullish coalescing.

■ Optional chaining (?.) safely accesses properties of possibly null or undefined objects. Nullish coalescing (??) provides a default value only when the left-hand side is null or undefined, not falsy.