UnCodemy

TypeScript Interview Questions PDF

TypeScript is JavaScript with static types that are checked before the code runs. Interviews ask about basic types, interfaces versus type aliases, union and intersection types, generics, type narrowing, utility types, any versus unknown, enums and the tsconfig settings. This page gives short model answers with small code examples.

Key takeaways

- TypeScript adds compile-time types and compiles to JavaScript.
- Interfaces and type aliases both describe shapes, with a few differences.
- Unions, narrowing and generics are the most tested ideas.
- Prefer unknown over any, and turn on strict mode.
- Types are erased at runtime, so they do not validate external data.

What is TypeScript interview questions?

TypeScript is a typed superset of JavaScript that adds static type checking and is compiled to plain JavaScript, helping developers catch errors earlier and write more maintainable code.

What is TypeScript and why use it?

TypeScript is a superset of JavaScript, so any valid JavaScript is valid TypeScript. It adds a type system checked at compile time. Benefits are earlier error detection, better editor help such as autocomplete and safer refactoring in larger code bases. The cost is extra syntax and a build step.

What are the basic types?

`string`, `number`, `boolean`, `null`, `undefined`, `bigint`, `symbol`, arrays (`number[]`), tuples (`[string, number]`), `object`, `any`, `unknown`, `void` and `never`. Types are usually inferred, so you can often omit annotations.

```
let age: number = 21;
let names: string[] = ["Asha", "Ben"];
let pair: [string, number] = ["id", 7];
```

What is the difference between any, unknown and never?

`any` turns off checking, so avoid it. `unknown` is a safe top type: you must check the type before using the value. `never` means a value that cannot occur, for example a function that always throws.

```
function parse(input: unknown) {
  if (typeof input === "string") return input.toUpperCase();
  return "";
}
```

Interface versus type alias?

Both describe object shapes. Interfaces can be extended and merged by declaring them again. Type aliases can name unions, intersections, tuples and primitives. For object shapes either works; follow your team's convention.

```
interface User { id: number; name: string; email?: string }
type Id = string | number;
```

What are union and intersection types?

A union (`A | B`) allows one of several types. An intersection (`A & B`) combines types so a value has all members.

```
type Result = "ok" | "error";
type Admin = User & { role: "admin" };
```

What is type narrowing?

Narrowing refines a union to a specific type inside a branch, using `typeof`, `instanceof`, the `in` operator, equality checks or a custom type guard. Discriminated unions use a shared literal field.

```
type Shape = { kind: "circle"; r: number } | { kind: "square"; side: number };
function area(s: Shape) {
  return s.kind === "circle" ? Math.PI * s.r ** 2 : s.side ** 2;
}
```

What are generics?

Generics let a function or type work with many types while keeping type safety.

```
function first<T>(items: T[]): T | undefined { return items[0]; }
const n = first([1, 2, 3]);   // number | undefined
```

You can constrain a generic: `<T extends { id: number }>`.

What are common utility types?

`Partial<T>` makes all properties optional, `Required<T>` makes them required, `Pick<T, K>` selects some, `Omit<T, K>` removes some, `Readonly<T>` blocks changes and `Record<K, V>` builds an object type. Know one example for each.

What do readonly, optional chaining and non-null assertion do?

`readonly` prevents reassignment. Optional chaining (`a?.b`) safely reads through null or undefined. The non-null assertion (`value!`) tells the compiler a value is not null; use it sparingly because it can hide bugs.

What is in tsconfig.json?

Options include `target`, `module`, `strict`, `noImplicitAny`, `esModuleInterop`, `outDir` and `include`. Turn on `strict` for new projects.

Do types protect me at runtime?

No. Types are erased in the compiled JavaScript. Data from an API or user input must be validated at runtime, for example with a schema validation library, before you trust its type.

What coding questions should I expect?

Common exercises are typing a function with a generic parameter, narrowing a union with a type guard, creating a utility type such as a readonly copy of an object, and fixing a type error in a short snippet. Practise by writing small examples in the TypeScript playground, and explain why the compiler reports each error. Reading error messages carefully is a skill interviewers like to see.

How does TypeScript relate to frameworks?

Frameworks such as React, Angular and Node-based back ends support TypeScript. In React you type props and state, in Angular TypeScript is the default language and in Node you add type definitions for libraries. Know how to install type packages, how to type an event handler and how to type API response data, and remember that runtime validation is still needed.

What best practices should I mention?

Enable strict mode, avoid any, prefer unknown for uncertain data, keep types close to where they are used, export shared types from one place, use discriminated unions for states such as loading, success and error, and let inference work instead of annotating every variable.

Frequently asked questions

What is TypeScript?

TypeScript is a typed superset of JavaScript that adds static type checking and compiles to plain JavaScript.

What is the difference between any and unknown?

any disables type checking, while unknown requires you to check the type before using the value, so it is safer.

Interface or type alias?

Both describe shapes. Interfaces can be extended and merged, while type aliases can also name unions and intersections.

Does TypeScript check types at runtime?

No. Types are removed when compiled, so external data must be validated at runtime.

What are generics used for?

Generics let functions and types work with many types while keeping type safety, such as a function returning the first item of any array.

Sources and further reading

- TypeScript documentation
- MDN JavaScript reference

About this PDF

These PDFs are study material for learning and revision. They are not a certificate, a degree or a promise of a job, and technology changes, so check official documentation for the latest details. Spot an error? Write to info@uncodemy.com and the trainer team will review it.

More free resources and courses: uncodemy.com | info@uncodemy.com | +91 98183 66550