CERTLABZ CERTIFICATION BOARD

Certified TypeScript Programmer (CTSP)

Ten Sample Questions

These ten questions are written at the difficulty of the live exam. Six carry a figure, a data table, a configuration or a code listing; four are plain scenario questions, as on the live paper. Every one is a scenario a working TypeScript programmer would meet, and none of them appears in the live question bank.

The live exam is multistage adaptive: how you score on one stage selects the difficulty of the next, so no two candidates sit the same paper. It is a performance-based exam: each form can carry up to 5 to 12 performance-based questions alongside the multiple-choice questions, which means you build, sequence or complete something rather than only choosing from a list. Answers and full explanations follow each question here; the live exam does not disclose item-level answers.

Question 1

Domain 1: TypeScript Fundamentals and Syntax

CONFIG

Emma Wilson compiles this file with tsc under the default settings.

```
let total: number = 5;
total = "ten";
```

Listing 1.1 A number reassigned to a string.

What does tsc report?

A. Compiles cleanly
B. Type error at tsc
C. Silent coercion
D. Prints a warning only

Correct answer: B

The variable total is annotated as a number, so assigning a string value fails the static type check and tsc reports a type error. TypeScript does not silently coerce the value or downgrade the mismatch to a warning under the default settings.

Question 2

Domain 2: Types, Interfaces and Enums

DATA

James Carter maps each construct to what it provides, with one entry missing.

Construct	Provides
interface	Shape of an object
enum	(to choose)
type alias	A name for any type
union	One of several types

Table 2.1 Constructs and what they provide.

What does an enum provide?

A. Named constants
B. A single private field
C. A loop counter
D. A generic bound

Correct answer: A
An enum defines a set of named constants that group related values under one type. It is not a private field, a loop counter, or a constraint on a generic, which are unrelated language features.

Question 3

Domain 3: Functions and Control Flow

CONFIG

Olivia Davis defines a function with an optional second parameter and calls it once.

```
function add(a: number, b?: number) {
  return a + (b ?? 0);
}
add(5);
```

Listing 3.1 An optional parameter with a fallback.

What does add(5) return?

A. NaN
B. undefined
C. 5
D. A compile error

Correct answer: C
The parameter b is optional, so calling add(5) leaves b as undefined and the nullish coalescing operator supplies 0. The function then returns 5 plus 0, which is 5, and no compile error occurs.

Question 4

Domain 4: Classes, Modules and Namespaces

CONFIG

Liam Anderson marks a field private and then reads it from outside the class.

```
class Box {
  private size = 4;
}
const b = new Box();
console.log(b.size);
```

Listing 4.1 Reading a private field.

What happens?

A. Prints 4
B. Prints undefined
C. Runtime crash
D. Blocked by tsc

Correct answer: D
A private field is only reachable inside the declaring class, so reading b.size from outside is blocked by tsc as a compile error. The access is rejected before any code runs, so it never prints a value or crashes at runtime.

Question 5

Domain 5: Generics and Advanced Types

DATA

Ava Thompson maps each utility type to its effect, with one effect missing.

Utility type	Effect
Partial<T>	All props optional
Readonly<T>	(to choose)
Required<T>	All props required
Pick<T, K>	Keep chosen keys

Table 5.1 Utility types and their effect.

What does Readonly<T> do?

A. Removes all keys
B. Locks each key
C. Adds new keys
D. Renames keys

Correct answer: B
Readonly<T> produces a type whose properties cannot be reassigned, so it locks each key against later writes. It does not remove, add, or rename keys, which are the jobs of other mapped and utility types.

Question 6

Domain 6: Tooling, Configuration and Best Practices

DATA

David Brown maps each tsconfig option to its effect, with one effect missing.

Option	Effect
strict	Enables strict checks
noImplicitAny	(to choose)
target	JS output version
outDir	Output folder

Table 6.1 Compiler options and their effect.

What does noImplicitAny do?

A. Requires types
B. Sets the target
C. Hides all errors
D. Formats the code

Correct answer: A
With noImplicitAny turned on, the compiler flags any value that would otherwise fall back to the any type, so it effectively requires explicit types. It does not set the output target, format code, or suppress errors.

Question 7 Domain 3: Functions and Control Flow

A function must accept either a number or a string and return a matching type, but a single loose signature types both as any and callers lose safety. What best restores type safety?

A. Use any everywhere
B. Remove the types
C. Cast to unknown
D. Declare overloads

Correct answer: D
Function overloads let you declare separate call signatures so each input type maps to the right return type while callers keep full checking. Widening to any, removing types, or casting to unknown all discard the safety the overloads restore.

Question 8 Domain 5: Generics and Advanced Types

A helper returns the first element of an array but is typed to return any, so every caller loses the element type. What best fixes this while keeping one implementation?

A. Return type any
B. Make it generic
C. Cast every call

D Return only unknown

Correct answer: B
Making the helper generic over the element type lets the return type flow from the array the caller passes, so each call stays fully typed. Returning any or unknown, or casting at every call site, either loses the type or pushes work onto callers.

Question 9 Domain 2: Types, Interfaces and Enums

A status field is typed as string, so typos such as activ pass the compiler and cause bugs at runtime. What best constrains it to the allowed values?

A A literal union
B Keep it a string
C The any type
D A wide number code

Correct answer: A
A union of string literal types lists exactly the allowed values, so the compiler rejects any status outside that set, including typos. Leaving it as a plain string, using any, or switching to an unbounded number all keep the field too wide to catch the error.

Question 10 Domain 6: Tooling, Configuration and Best Practices

A team disables strict mode to silence compiler complaints, and null related bugs soon reach production. What is the sound practice?

A Disable all checks
B Ignore null cases
C Keep strict on
D Ship without tsc

Correct answer: C
Keeping strict mode on turns on strict null checks and related safeguards that catch the very bugs the team let through. Disabling checks, ignoring null cases, or skipping tsc removes the safety net rather than fixing the underlying code.

Published by the CertLabz Certification Board, a division of CertLabz LLC. Companion to objectives document CTSP-001. https://certlabz.com/certifications/certified-typescript-programmer.html