📘Chapter 11: Type Guards & Exhaustiveness Checks

***What is a Type Guard in TypeScript?***

A Type Guard is a technique in TypeScript that checks the type of a value at runtime and helps the compiler narrow down the type in a safe way.

It tells TypeScript "Hey, inside this block, I’m sure this is type X - not Y."

***Why Do Type Guards Exist?***

Because TypeScript doesn’t always know the exact type of a union or unknown value.  
You use a Type Guard to

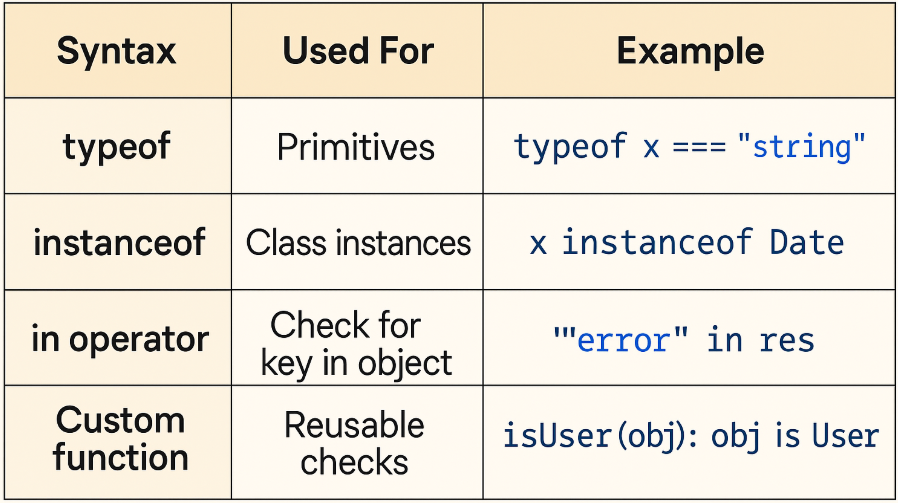
1. ✅ Narrow types safely.
2. 🚫 Avoid runtime errors.
3. 📦 Access the correct properties/methods confidently.

***Real-Life Analogy***

Imagine someone hands you a sealed box.  
You don’t know if it’s a laptop 📱 or a book 📖.  
Before using it, you must open and check. That’s your type guard!

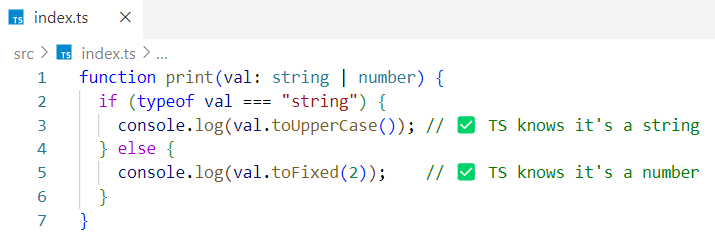
***Built-in Type Guards***

TypeScript has 4 main type guard techniques:

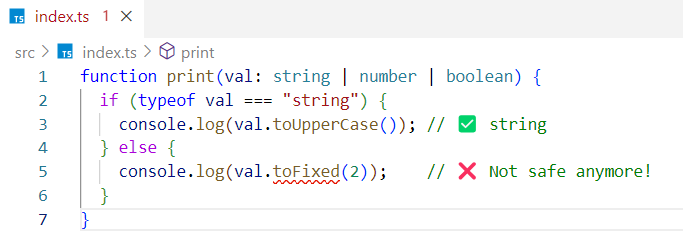


***Code Examples (with inline explanation)***

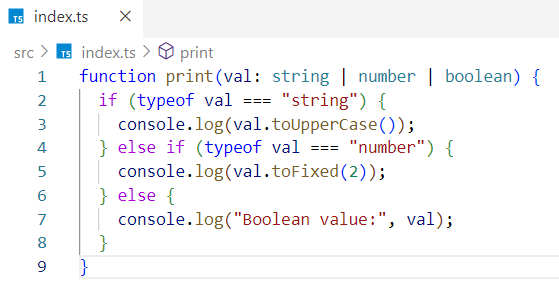
**1️. typeof - For primitive types**



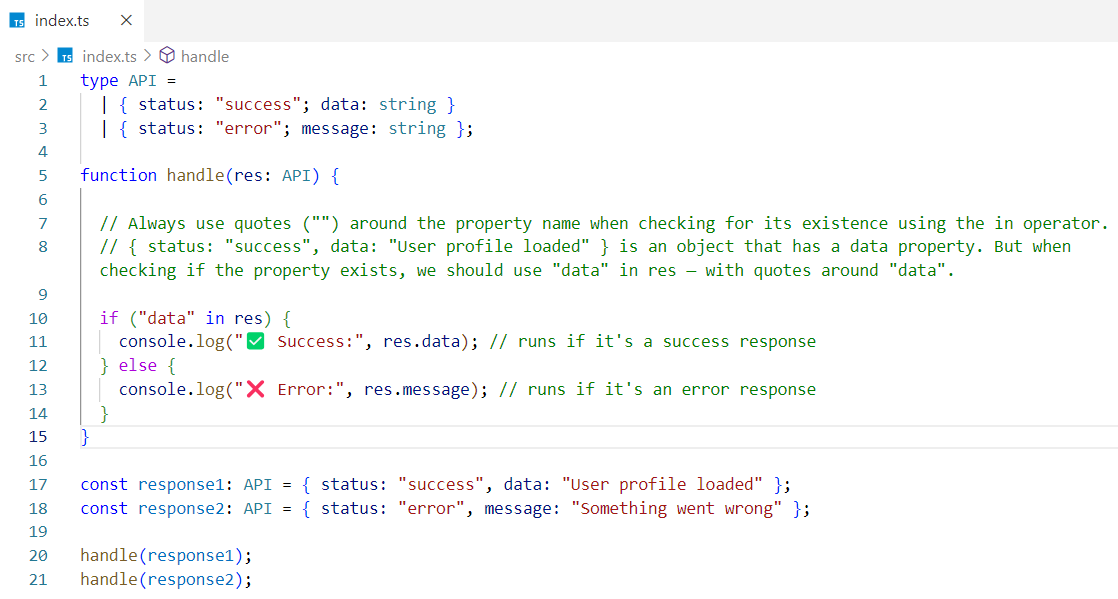
When a union has only **two types,** checking one type lets TypeScript automatically assume the other in the else block. But when the union has **more than two types,** you must explicitly check for **each type** - the else block is no longer safe by default.



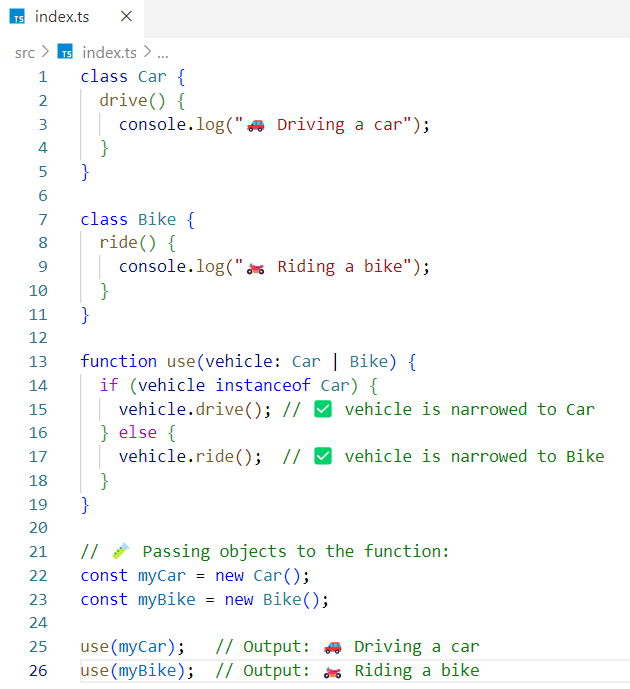
Fix -



**2️. in operator - For objects**



**3️. instanceof - For classes**



With object types, we check the shape of the object;

with class types, we use instanceof to check if the object was created from a specific class.

**4️. Custom Type Guard Function**

* A **custom type guard** is a user-defined JavaScript function that returns a **boolean value** from a conditional expression.
* Its **return type** is a special **type predicate** in the form <value> is <Type>.
* If the function returns true, TypeScript understands that the value **is of the specified type**.  
  If it returns false, TypeScript treats it as **not of that type.**
* In our case, the value is the person object, and the possible Type can be User, Admin, or SuperAdmin.

In the example below, if you try to access person.name or person.accessLevel directly inside the function (outside any condition),TypeScript will show an error because it doesn’t yet know whether person is a User or an Admin. That’s why you need to narrow the type first using either a custom type guard, or a type check like if (person.role === "user").



If the isUser function returns true, then TypeScript will treat person as a User from that point onward.

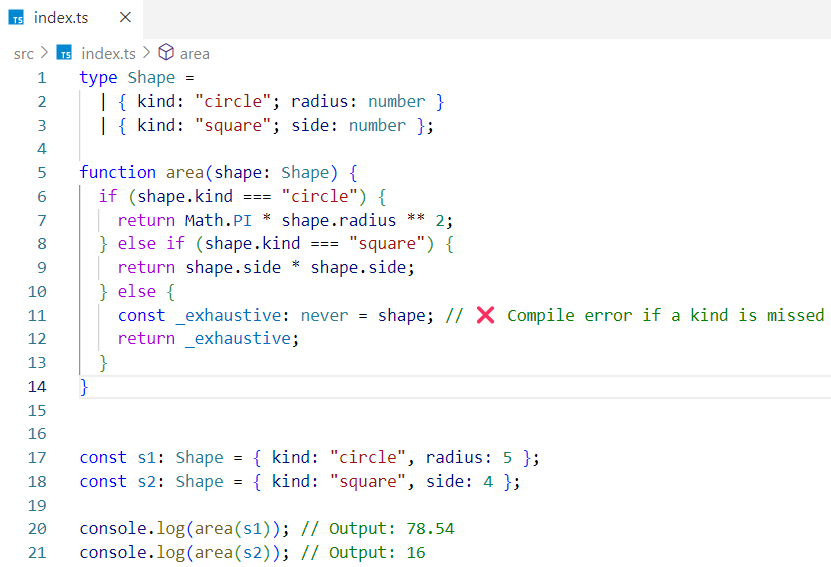
The same applies to other custom type guard functions - they help TypeScript understand the exact type when the condition is true.

Note - The **condition inside a type guard function** is usually based on the **discriminant tag** - like role, status, kind, etc.

***How Type Guards Connect to Exhaustiveness Checks ?***

Type Guards help narrow the type, and **never** ensures you don’t miss a case.

Example:



Interview Insights

***Q1: What is a Type Guard?***

A Type Guard is a runtime check that tells TypeScript the specific type of a variable inside a block.

***Q2: What are the different ways to write type guards?***

* typeof for primitives
* in for object keys
* instanceof for classes
* Custom functions using x is Type

***Q3: How is it different from regular if checks?***

Regular if checks don’t narrow types safely — type guards do and inform the compiler what’s safe to access.