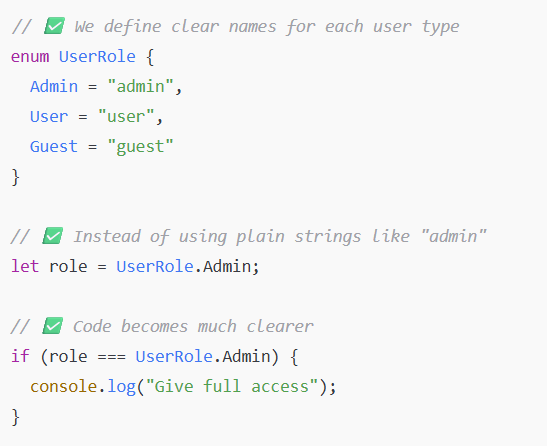
Chapter 9 - Enums

***What is an Enum?***

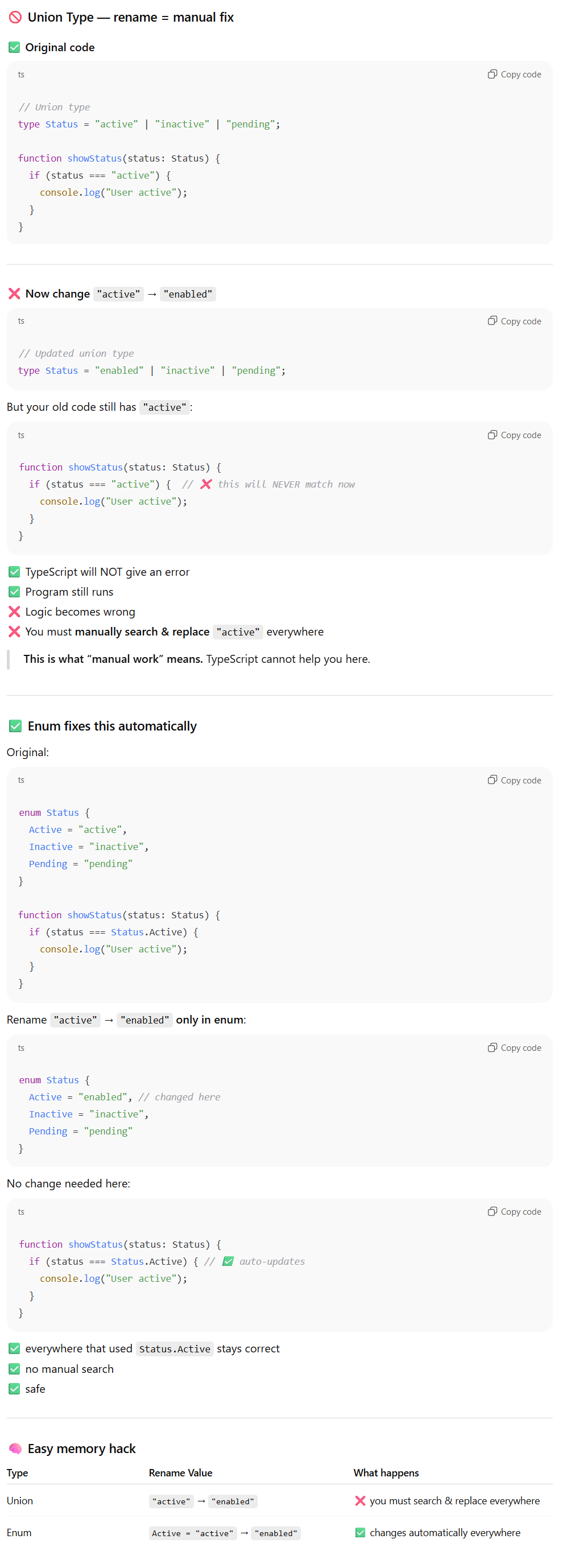
An enum (short for "enumeration") is a special TypeScript feature that lets you define a set of named constant values.

✅ Enum makes your code more readable by giving meaningful names to fixed values  
✅ Enum is Ideal for representing things like user roles, app states, modes, and status

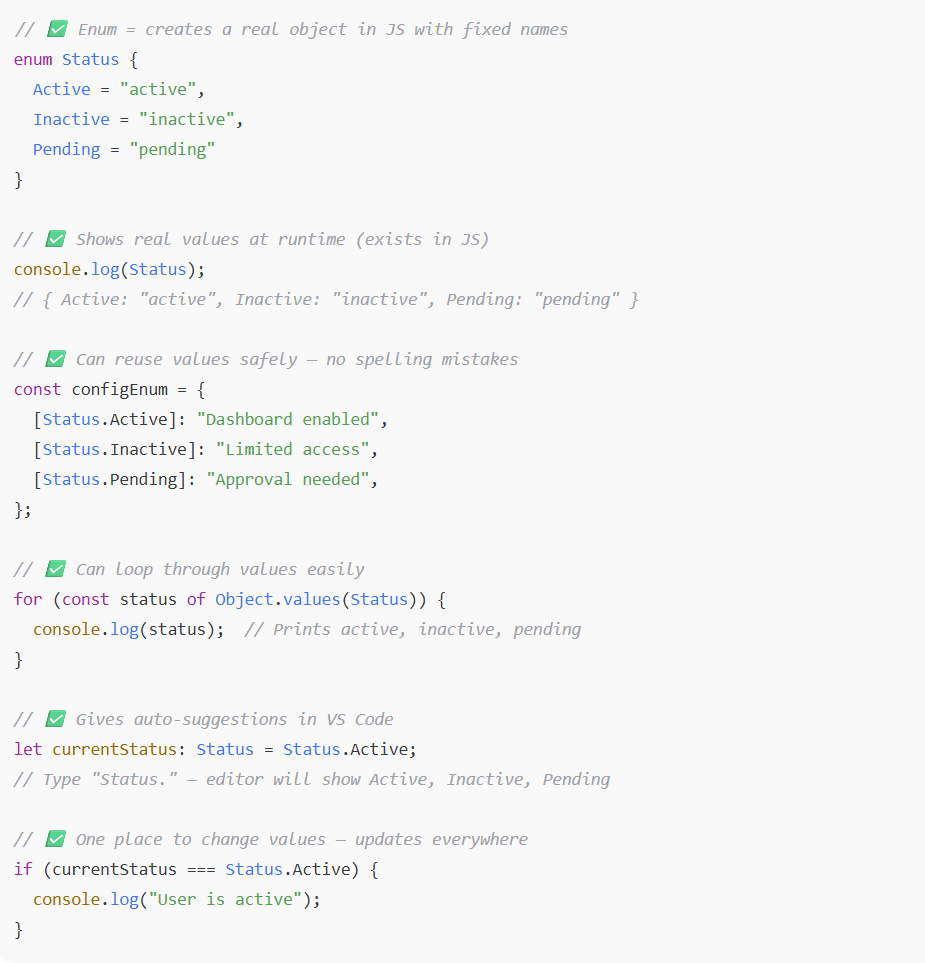


***Example - Without Enums – (using literal Union Types)***



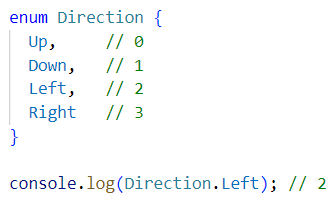


***Example - With Enums –***



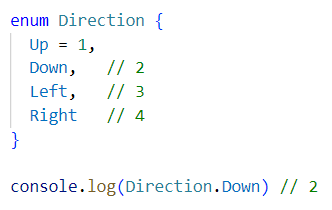
***One place to change values means where we define enums that’s the only place where we can change value once as per the requiremenst .***

***Numeric Enums (Default) -***

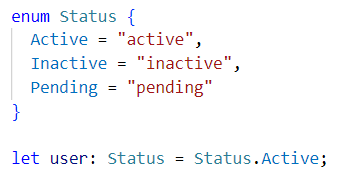


Defaults to 0 and auto-increments by 1 for each subsequent member.

You can manually set values -

  
  
***String Enums -***

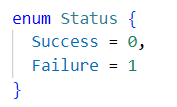
Use when you want readable values instead of numbers.



❌ Not auto-incremented

***What is Reverse Mapping in TypeScript?***

Reverse mapping means you can use a value to look up its corresponding enum key.

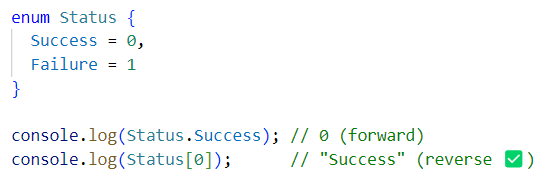


In normal objects, you do obj.key and get the value specific to that key.

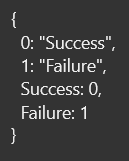
In numeric enums, TypeScript gives you both:

* Status.Success → 0 ✅ // forward
* Status[0] → "Success" ✅ // reverse mapping

### Real Example - Reverse Mapping (only for numeric enums) -



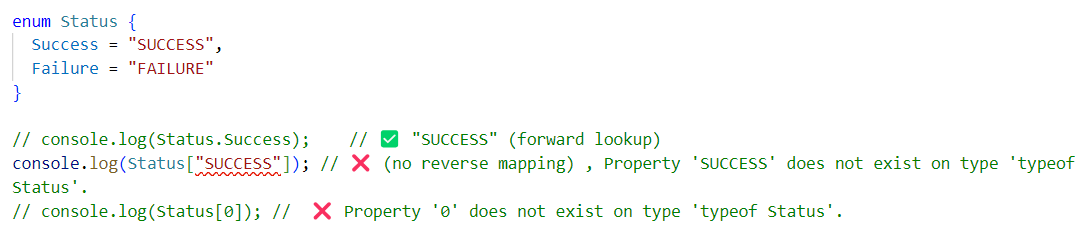
TypeScript builds this under the hood -

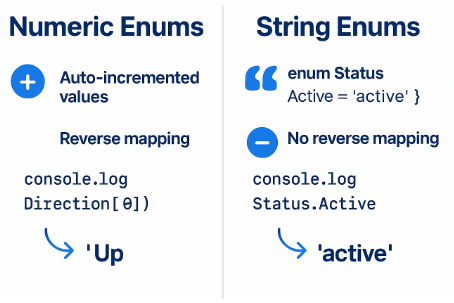


So, you can access both ways -

* ✅ Status.Success → 0
* ✅ Status[0] → "Success"

### ❌ Reverse mapping does not work in String Enums





✅ Use Case: API Response Status with Enums



### *Does this mean the API response must have all 3 statuses?*

No. It means: The status you receive must be one of the 3 defined in the enum — nothing more, nothing less.

You’re not requiring all 3 at once.  
You're saying: “the status parameter must be one of these valid values”:

* "success"
* "error"
* "timeout"

Note - In TypeScript, when using object types, you must define all expected properties and their types - partial or incomplete shapes will cause errors unless marked optional.

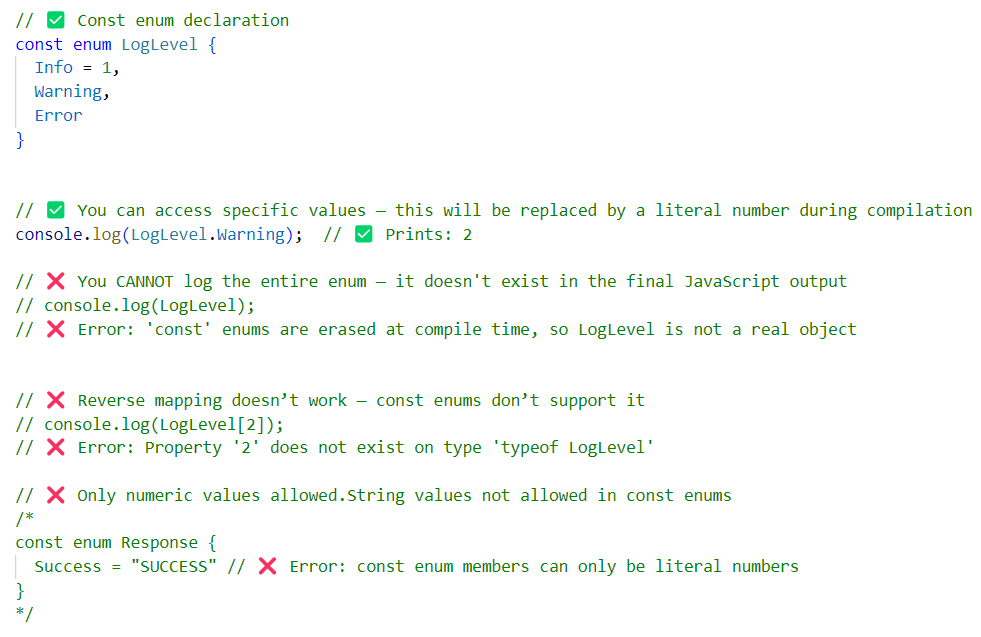
***Const Enums -***

### What is a const enum?

A const enum is a special type of enum in TypeScript that is -

✅ Super lightweight — improves performance and reduces bundle size. Means A normal enum creates a big JS object at runtime (takes up space in your JS file ✅).But a const enum removes the object entirely and just replaces values inline:

❌ Does not support reverse mapping  
❌ Only supports numeric values



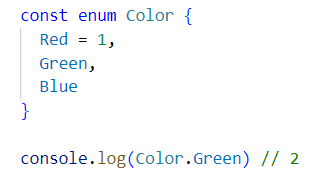
Example 1 -

### 

### ✅ Explanation -

* Color.Red → inlined as 0
* No Color object is created at runtime
* You cannot do: Color[0] ❌ (no reverse mapping)

Example 2 –



### ✅ Explanation -

* Color.Red = 1 → You manually assigned 1
* Color.Green → Auto-incremented from Red → becomes 2
* Color.Blue → Auto-incremented → becomes 3
* console.log(Color.Green) → ✅ Inlined as 2 at compile time
* ❌ No Color object created at runtime
* ❌ Reverse mapping like Color[2] won't work — it's not available for const enums

Interview Questions

***Q1: What’s the difference between numeric and string enums?***

Numeric enums auto-increment and support reverse mapping; string enums don’t, but are human-readable.

***Q2: When should you use const enum?***

When you want performance + minimal output JS — but only with numeric values.

***Q3: Can enums replace union string types?***

They can, but union types are more tree-shakable and readable for external APIs.

***Q4: Are enums available at runtime?***

Yes - enums are compiled into JS objects.

### *Q5: What does “tree-shakable” mean?*

"Tree-shaking" is a build optimization technique (used by tools like Webpack, ESBuild, Vite) that removes unused code from your final bundle.

#### **Union Types = Tree-shakable**

// You define a type — not a real JS object

type Status = "active" | "inactive";

// No JS code is generated, so it's easy to eliminate unused code.

✅ Nothing gets included in the final bundle unless it's used.

#### **Enums = NOT Tree-shakable**

enum Status {

Active = "active",

Inactive = "inactive"

}

// This generates a real JS object in your bundle - even if you don’t use all members. That object can't be tree-shaken out by default.

### *Q6: Why unions are better for external APIs ?*

Imagine you're modelling responses from an API.

type APIStatus = "success" | "error" | "timeout"; // ✅ Matches the API response exactly

// Compare with-

enum APIStatus {

Success = "success",

Error = "error",

Timeout = "timeout"

}

Why because -

**Matches API data directly**  
The server sends plain strings like "success" — union types accept them as-is with no conversion.

**No extra mapping needed**  
You don’t need to compare or translate values like APIStatus.Success; you use the exact string returned by the API.

**Lighter and faster**  
Union types don’t create JavaScript code — they only exist in TypeScript, so they add zero runtime cost.

**Perfect for JSON/API responses**  
APIs return raw strings in JSON — unions fit naturally with that format.

**Less code, cleaner logic**  
No enums, no extra comparisons — your code becomes simpler and easier to read.



