

1. The Basics: Traffic Light System

Goal: Create a simple enum and use it in a switch statement.

- **Task:** Define an enum called `TrafficLight` with values: Red, Yellow, and Green.
- **Logic:** Write a method `GetAction(TrafficLight light)` that returns a string:
 - Red → "Stop"
 - Yellow → "Caution"
 - Green → "Go"
- **Bonus:** Add a default case that handles an undefined enum value.

2. Enums with Custom Integer Values

Goal: Understand that enums are backed by integers.

- **Task:** Create an enum `HttpError` where:
 - `BadRequest` = 400
 - `Unauthorized` = 401
 - `NotFound` = 404
 - `InternalServerError` = 500
- **Logic:** Ask the user to input an integer. Cast that integer to the `HttpError` enum and print the name of the error. If the number doesn't match a defined error, handle it gracefully.

3. Bitwise Flags: File Permissions

Goal: Use the [Flags] attribute to combine multiple enum values.

- **Task:** Define an enum Permission with the [Flags] attribute.
 - Values: None = 0, Read = 1, Write = 2, Execute = 4, Delete = 8.
- **Logic:** 1. Create a variable myPermissions that combines Read and Write.

2. Write code to check if myPermissions has the Write flag.

3. Add the Execute flag to the variable and print the result.

4. Parsing Strings to Enums

Goal: Handle external data (like API responses or user input) safely.

- **Task:** Create an enum Difficulty (Easy, Medium, Hard).
- **Logic:** Take a string input from the user. Use `Enum.TryParse<Difficulty>(input, out var result)` to convert the string to the enum.
 - If successful, print "Difficulty set to [result]".
 - If failed, print "Invalid difficulty level."

5. Iterating and Descriptions

Goal: Use Enum.GetValues to automate logic.

- **Task:** Create an enum DaysOfWeek.
- **Logic:** Use a foreach loop to iterate through every value in the DaysOfWeek enum.
For each value:
 1. Print the integer value.
 2. Print the name.
 3. Check if the day is Saturday or Sunday and print "(Weekend)" next to it.