Assignment Objective

Design, plan, and implement a program that evaluates a user's proposed password against a set of security rules, and gives a score of the password strength.

Core Tasks

Part 1: Problem Analysis & Design

1. Understand the Rules:

Your evaluator must check a candidate `String` password for the following criteria:

- The length of the password.
- Whether it contains any uppercase letters (`A-Z`).
- Whether it contains any lowercase letters (`a-z`).
- Whether it contains any digits (`0-9`).
- Whether it contains any special character from this set: `! @ # $ % & * ?`.

Determine a scale for the strength of password, perhaps a scale from zero to five? Whatever you decide, remember your reasons, and clearly document the scale when writing your documentation for the project.
Your code must take a string password and return a value representing the strength of the password based on the scale you've decided on.

2. Decompose the Problem (then use Abstraction when designing the API):

To clearly and efficiently implement the project, you will need to break down the problem into smaller problems that will be easy to code and test. It is left to the student to decide how to decompose the problem.

4. Documentation:

Draw flowcharts that will allow a reader to easily understand the password evaluation process. If you've done your decomposition and abstraction well, this will be an easy task. I expect you will only need two or three simple flowcharts. You do not need to draw a flowchart for any subprocess that would have a very obvious implementation.

Part 2: Implementation and Testing

Determine ways to reduce duplication of code, such as when you notice repeated code, replace it with a more generic function that takes parameters for the specific differences. As you become a more experienced programmer, you will become better at recognizing where code will be re-used, and writing the generic functions before writing the duplicated code.

You may also wish to try a "multi-pass" approach, meaning that after you have written some or all of the program, you may realize there's a cleaner, more elegant way to implement the code. Don't be afraid to start from scratch and re-write all or some of the code. In many cases, it will actually save you time and effort over trying to fix messy code.

For testing, you may wish to first input a user password and output the calculated strength. However, in case the code needs to be updated at a later time, you should also write code that will test a list of predetermined passwords and compare them to the expected strength according to your

documented strength scale. Meaning: for each password string you will test, you will likely want to provide the associated strength value that the code is expected to return. You can verify the value returned from the password evaluator is the same as the expected value. In this way, with proper test code, if the password validator code is modified, re-running the test code will automatically provide a variety of cases to ensure the updated code still operates according to specifications.

Part 4: Going Beyond

You can optionally do these additional mini-projects related to this project:

- We will still need to practice coding how to read from and write to a file in Python. For your test method, you could provide a list of passwords to test in a file, and load that password test file into your test code rather than hard coding passwords to test in the test code.

Minimum Completion Checklist

✅ Clear documentation for the strength scale.

✅ Complete flowcharts that make the overall password evaluator algorithm easily understandable.

✅ Password evaluator Python code that operates correctly.

✅ Test code that is clean and tests the code reasonably thoroughly.

✅ All code is neatly formatted, and uses meaningful variable names