

## PASSWORD COMPLEXITY CHECKER

CHECK OUT THIS PROJECT ON MY GITHUB : [https://github.com/sajilsaju/PRODIGY\\_CS\\_03.git](https://github.com/sajilsaju/PRODIGY_CS_03.git)

### REPORT SUBMITTED BY



**SAJIL MOHAMED V**



**+91 8714194649**



[sajilmohamed1234@gmail.com](mailto:sajilmohamed1234@gmail.com)



<https://www.linkedin.com/in/sajilmohamed1718/>



<https://github.com/sajilsaju/>

<b>1. Introduction</b>	→	<b>2</b>
<i>1.1 Project Overview</i>		
<b>2. Objectives</b>	→	<b>2</b>
<b>3. Project Design and Implementation</b>	→	<b>3</b>
<i>3.1 Criteria for Password Strength</i>		
<i>3.2 Implementation Details</i>		
<i>3.3 Program Code</i>	→	<b>4</b>
<b>4. Results And Discussion</b>	→	<b>6</b>
<b>5. Knowledge Gained</b>	→	<b>7</b>
<b>6. Challenges Faced</b>	→	<b>8</b>
<b>7. Conclusion</b>	→	<b>9</b>
<b>8. Future Work</b>	→	<b>9</b>
<b>9. Reference</b>	→	<b>10</b>

PRODIGY INFOTECH

# 1. Introduction

## 1.1 Project Overview

Passwords are a critical component of cybersecurity, serving as the first line of defense against unauthorized access to sensitive data. The increasing complexity of cyber threats necessitates strong passwords that are difficult to crack. However, users often create weak passwords that are easy to remember but also easy to compromise. To address this issue, a Password Complexity Checker tool has been developed to assesses the strength of a password based on a criteria such as length, presence of uppercase and lowercase letters, numbers, and special characters. And also provide feedback to the users on the password's strength.

## 2. Objective

The objective of this project is to develop a Python-based Password Complexity Checker that evaluates the strength of a password based on predefined criteria. The tool categorizes passwords into three levels: **Strong**, **Moderate**, and **Poor**, and provides actionable feedback to help users create more secure passwords.

### Features:

- **Password Strength Evaluation:** Classifies passwords into Strong, Moderate, or Weak categories.
- **Criteria-Based Analysis:** Checks for length, uppercase and lowercase letters, numbers, and special characters.
- **User Feedback:** Provides actionable suggestions for creating stronger passwords.
- **Simple and Lightweight:** Easy-to-use command-line interface for quick assessments.

## 3. Project Design And Implementation

### 3.1 Criteria for Password Strength

The Password Complexity Checker evaluates passwords based on the following criteria:

- **Length:** A minimum of 8 characters.
- **Uppercase Letters:** Presence of at least one uppercase letter.
- **Lowercase Letters:** Presence of at least one lowercase letter.
- **Numbers:** Inclusion of at least one numeric digit.
- **Special Characters:** Presence of at least one special character (.,=,!@#\$%^&\*(),.?:{}|<>).

### 3.2 Implementation Details

The tool is implemented in Python and utilizes the re module for regular expression matching. The program works as follows:

- **User Input:** The user is prompted to enter a password.
- **Criteria Evaluation:** The password is evaluated against the five criteria listed above.
- **Strength Classification:** Based on the number of criteria met, the password is classified as **Strong** (all criteria met), **Moderate** (3 or 4 criteria met), or **Poor** (fewer than 3 criteria met).
- **Feedback:** If the password is weak, the tool provides specific recommendations to improve its strength.

### **3.3 Program Code**

```
import re

def check_password_strength(password):

    # Define the criteria for password strength

    length_criteria = len(password) >= 8

    uppercase_criteria = re.search(r"[A-Z]", password) is not None

    lowercase_criteria = re.search(r"[a-z]", password) is not None

    number_criteria = re.search(r"[0-9]", password) is not None

    special_char_criteria = re.search(r"[!@#$%^&*(),.\?\"':{}|<>]", password) is
not None


    # Calculate strength based on the criteria

    strength = sum([length_criteria, uppercase_criteria, lowercase_criteria,
number_criteria, special_char_criteria])


    # Provide feedback based on the strength

    if strength == 5:

        return "Strong!!"

    elif 3 <= strength < 5:

        return "Moderate!"

    else:

        feedback = "Poor!. Consider the following improvements:\n"

        if not length_criteria:
```

```
feedback += "- Ensure the password is at least 8 characters long\n"
```

```
if not uppercase_criteria:
```

```
feedback += "- Include at least one uppercase letter\n"
```

```
if not lowercase_criteria:
```

```
feedback += "- Include at least one lowercase letter\n"
```

```
if not number_criteria:
```

```
feedback += "- Include at least one digit\n"
```

```
if not special_char_criteria:
```

```
feedback += "- Include at least one special character  
(!@#$%^&*(),.?\"':{}|<>)\n"
```

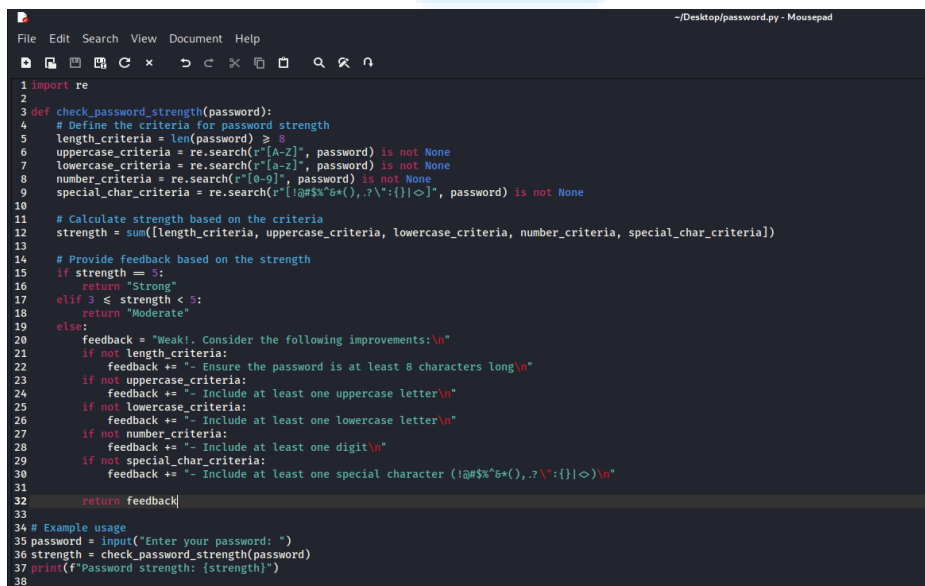
```
return feedback
```

```
# Example usage
```

```
password = input("Enter your password: ")
```

```
strength = check_password_strength(password)
```

```
print(f"Password strength: {strength}")
```

A screenshot of a code editor window titled "password.py - Mousepad". The editor shows a Python script for checking password strength. The script defines a function `check\_password\_strength` that takes a password as input and returns a strength value and feedback. The strength is calculated based on four criteria: length (at least 8 characters), uppercase letters (at least one), lowercase letters (at least one), and digits (at least one). The feedback is a string of messages indicating which criteria are not met. The script also includes an example usage section at the bottom.

```
1 import re
2
3 def check_password_strength(password):
4     # Define the criteria for password strength
5     length_criteria = len(password) >= 8
6     uppercase_criteria = re.search(r"[A-Z]", password) is not None
7     lowercase_criteria = re.search(r"[a-z]", password) is not None
8     number_criteria = re.search(r"[0-9]", password) is not None
9     special_char_criteria = re.search(r"[!@#$%^&*(),.?\"':{}|<>]", password) is not None
10
11     # Calculate strength based on the criteria
12     strength = sum([length_criteria, uppercase_criteria, lowercase_criteria, number_criteria, special_char_criteria])
13
14     # Provide feedback based on the strength
15     if strength == 5:
16         return "Strong"
17     elif 3 <= strength < 5:
18         return "Moderate"
19     else:
20         feedback = "Weak!. Consider the following improvements:\n"
21         if not length_criteria:
22             feedback += "- Ensure the password is at least 8 characters long\n"
23         if not uppercase_criteria:
24             feedback += "- Include at least one uppercase letter\n"
25         if not lowercase_criteria:
26             feedback += "- Include at least one lowercase letter\n"
27         if not number_criteria:
28             feedback += "- Include at least one digit\n"
29         if not special_char_criteria:
30             feedback += "- Include at least one special character (!@#$%^&*(),.?\"':{}|<>)\n"
31
32     return feedback
33
34 # Example usage
35 password = input("Enter your password: ")
36 strength = check_password_strength(password)
37 print(f"Password strength: {strength}")
38
```

## 4. Results And Discussion

### Evaluation of Sample Passwords

Several sample passwords were tested using the tool to verify its functionality:

- **Password:** @#Sajilsaju123\_>

**Result:** Strong!!

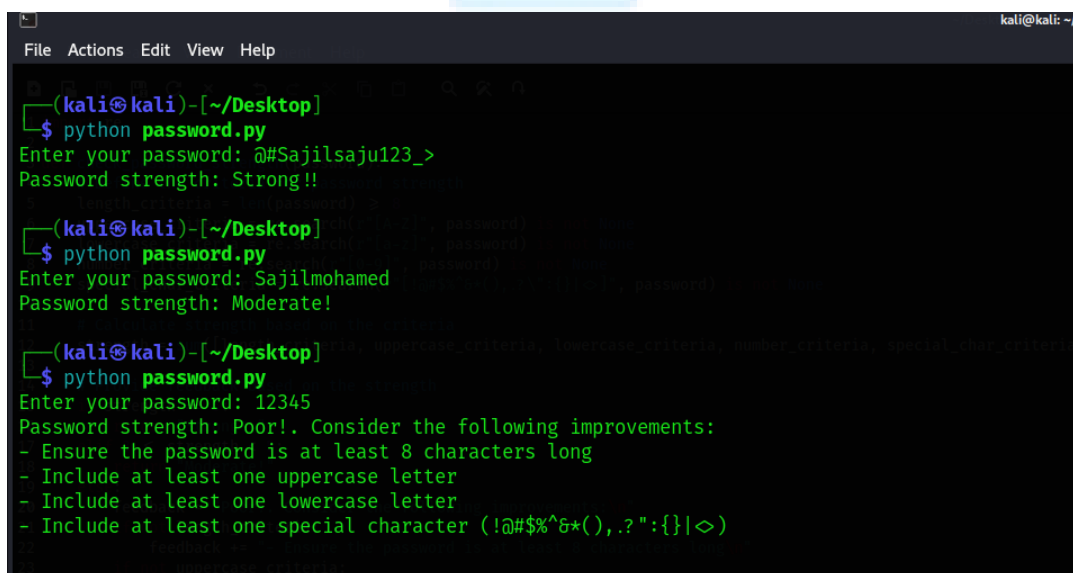
- **Password:** Sajilmohamed

**Result:** Moderate!

- **Password:** 12345

**Result:** Poor! Consider the following improvements:

- Ensure the password is at least 8 characters long
- Include at least one uppercase letter
- Include at least one lowercase letter
- Include at least one special character (!,@#\$%^&\*(),.?":{}|<>)



```
kali@kali: ~/Desktop
File Actions Edit View Help
(kali@kali)-[~/Desktop]
$ python password.py
Enter your password: @#Sajilsaju123_>
Password strength: Strong!!

(kali@kali)-[~/Desktop]
$ python password.py
Enter your password: Sajilmohamed
Password strength: Moderate!

(kali@kali)-[~/Desktop]
$ python password.py
Enter your password: 12345
Password strength: Poor!. Consider the following improvements:
- Ensure the password is at least 8 characters long
- Include at least one uppercase letter
- Include at least one lowercase letter
- Include at least one special character (!@#$%^&*(),.?":{}|<>)
```

## 5. Knowledge Gained

- **Deepened Understanding of Password Security:** Gained insight into the key criteria that make passwords secure, such as length, use of uppercase and lowercase letters, numbers, and special characters.
- **Regular Expressions Mastery:** Enhanced my skills in using regular expressions (re) in Python to validate and analyse password patterns efficiently.
- **User Feedback Mechanisms:** Learned the importance of providing actionable feedback to users, which is critical in guiding them towards better security practices.
- **Strength Grading Implementation:** Developed an approach to categorize passwords into different strength levels (Weak, Moderate, Strong), which can be applied to various cybersecurity tools.
- **Real-World Application of Python:** Strengthened my Python programming skills by applying them to solve practical cybersecurity challenges.
- **Project Structuring:** Improved my ability to structure and document a project, including creating comprehensive README files and ensuring code is user-friendly and maintainable.

PRODIGY INFOTECH



## 6. Challenges Faced

- I. **Balancing Criteria Sensitivity:** One of the key challenges was determining the right balance for the password strength criteria. Ensuring that the tool is neither too strict nor too lenient required careful consideration and testing.
- II. **Regular Expression Complexity:** Writing regular expressions that accurately capture the presence of various character types (uppercase, lowercase, digits, special characters) without false positives or negatives was challenging, especially when dealing with edge cases.
- III. **Feedback Generation:** Developing a system that provides clear, actionable feedback to users for improving their passwords was challenging. It required thinking from the user's perspective to ensure that the suggestions were easy to understand and implement.
- IV. **Handling Edge Cases:** Some passwords could meet criteria in unexpected ways, such as using symbols that resemble letters (e.g., '1' instead of 'l'). Ensuring the tool could handle these edge cases accurately was a non-trivial task.
- V. **Efficiency and Complexity:** Ensuring that the password checking tool was both efficient and comprehensive posed a challenge. Adding more complex checks without significantly impacting performance required optimization.

## 7. Conclusion

The development of the Password Complexity Checker has been a valuable exercise in understanding the importance of secure password creation and the challenges users face in generating strong passwords. The tool successfully evaluates password strength based on essential criteria, categorizing them as Strong, Moderate, or Weak, and provides constructive feedback to guide users in improving their passwords. This project not only reinforced fundamental concepts in cybersecurity but also highlighted the importance of user-friendly design in security tools. Moving forward, the tool can be expanded with additional features and integrated into broader security frameworks, making it a practical and robust solution for enhancing password security in various applications

## 9. Future Works

- ❖ **Integration with User Authentication Systems:** The tool can be integrated into user authentication systems, such as login and registration forms, to provide real-time password strength feedback during password creation.
- ❖ **Advanced Password Analysis:** Future versions could incorporate more advanced analysis, such as detecting commonly used passwords, dictionary words, or repeated patterns, which are often targeted in brute-force attacks.
- ❖ **Localization and Internationalization:** Enhancing the tool to support multiple languages and regional settings can make it more accessible to a global audience.

- ❖ **GUI Implementation:** Developing a graphical user interface (GUI) for the tool would make it more user-friendly and accessible to non-technical users.
- ❖ **Password Generation Feature:** Adding a secure password generator feature could help users create strong passwords directly within the tool, further enhancing its utility.
- ❖ **Machine Learning Integration:** Implementing machine learning algorithms to predict password strength based on historical password breach data could provide a more dynamic and robust assessment.
- ❖ **Mobile and Web Extensions:** Expanding the tool into a mobile app or browser extension could make it easier for users to check password strength on-the-go or as they browse the web.
- ❖ **Security Enhancements:** Implementing additional security measures, such as rate limiting or hashing of input data, could further protect the tool from misuse or attacks.

## 9. Reference

- **Python Regular Expressions Documentation:** Understanding and implementing regular expressions for pattern matching in password validation.
- **OWASP Password Strength Guidelines:** A comprehensive guide on best practices for creating secure passwords.
- **Stack Overflow Discussions:** Community insights and solutions on handling edge cases and improving regular expressions for password validation.
- **Common Password Patterns:** Studies and datasets on commonly used passwords and password patterns to avoid in secure password creation.

