

Project Title: Password Strength Checker

Course: Build Your Own Project – VITyarthi

Submitted By: Yogesh

Registration Number: 25BCY10076

Program: B.Tech

School: VIT Bhopal University

Date of Submission: 23 November 2025

Introduction

In today's digital world, passwords act as the primary line of defense for protecting user data and digital identities. However, weak or predictable passwords remain a major security risk, as attackers often exploit common patterns, simple structures, or dictionary words to gain unauthorized access.

This project presents a Password Strength Checker that evaluates the security level of a password based on several parameters such as length, character diversity, entropy, and common-password detection. The goal is to educate users on choosing safer passwords and to provide real-time feedback on improving password quality.

Problem Statement

Many users choose weak passwords that can be easily guessed or cracked using brute force, dictionary attacks, or social engineering.

There is a need for a simple yet effective system that can:

- *Detect weak passwords
- *Suggest improvements
- *Help users build stronger, more secure passwords

This project aims to solve this issue by developing a Password Strength Checker using Python.

Functional Requirements

1. The system must accept a password input from the user.

2. It must evaluate strength based on:

- *Length of password
- *Uppercase and lowercase characters
- *Numerical digits
- *Special characters
- *Presence in common-password database
- *Entropy score

3. The system must categorize the password as:

*Weak

*Medium

*Strong

*Very Strong

4. The system must display the reason behind the strength level (optional enhancement).

5. Non-Functional Requirements

1. Performance: Password evaluation must occur in less than 1 second.

2. Security: Passwords should not be saved, logged, or transmitted.

3. Usability: Simple, clean interface that anyone can use.

4. Reliability: Must provide consistent results for all inputs.

5. Maintainability: Code should be modular and easy to update.

System Architecture

Architecture Flow:

User Input → Input Module → Password Evaluation Engine
→ Entropy Calculator → Strength Decision Module → Output Result

Modules Used:

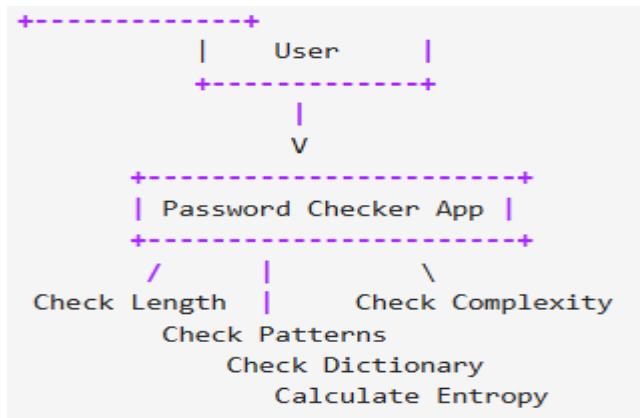
Input Handler
Pattern Checker
Entropy Calculator
Common Password Validator
Output Formatter

Design Diagrams

(Insert the diagrams after I generate them — tell me “Generate diagrams” and I will provide images.)

You must include the following:

1. Use Case Diagram



2. Workflow Diagram

```

START
↓
Input password
↓
Check length → Check character types → Check dictionary words → Check patterns → Calculate entropy
↓
Generate score
↓
Display rating + suggestions
↓
END

```

3. Sequence Diagram

User → System: Enter password

System → Length Module: Validate length

System → Complexity Module: Analyze characters

System → Dictionary Module: Check common words

System → Pattern Module: Detect sequences

System → Entropy Module: Calculate entropy

System → User: Provide rating + suggestions

4. Class / Component Diagram

```
+-----+
| PasswordChecker |
+-----+
| +check_length() |
| +check_complexity() |
| +check_dictionary() |
| +check_patterns() |
| +calculate_entropy() |
| +generate_score() |
+-----+
```

5. ER Diagram – (Not required since no database storage is used. Write “Not Applicable”)

USER_TABLE

User_ID (PK)
Password
Strength_Score
Timestamp

Design Decisions & Rationale

1. Python was chosen because it offers:

*Fast development

*Built-in regex libraries

*Easy entropy calculation

2. Entropy-based evaluation gives more accurate strength measurement than simple rule checking.

3. No database used to ensure user privacy.

4. Modular design ensures easy future upgrades (GUI, Mobile App, Website version etc.).

Implementation Details

Technologies / Tools Used

Python (Core Logic)

Regular Expressions

Math module (Entropy calculation)

Command-line interface

Code Structure

password_checker/

```
|  
|--- password_strength_checker.py  
|--- common_passwords.py (optional)  
└--- README.md
```

Main Logic Components

1. Pattern Checking – Detects uppercase, lowercase, digits, symbols
2. Entropy Calculation – Uses mathematical formula
3. Common Password Detection – List contains commonly used passwords
4. Output Module – Displays strength level

Python Code Used (Full)

```
import re  
  
import math  
  
common_passwords = ["password", "123456", "qwerty", "admin", "letmein"]  
  
def calculate_entropy(password):  
    pool = 0  
  
    if re.search(r"[a-z]", password): pool += 26  
    if re.search(r"[A-Z]", password): pool += 26
```

```
if re.search(r"[0-9]", password): pool += 10
if re.search(r"[^A-Za-z0-9]", password): pool += 32
return len(password) * math.log2(pool) if pool else 0

def check_strength(password):
    score = 0

    if len(password) >= 8: score += 1
    if re.search(r"[A-Z]", password): score += 1
    if re.search(r"[a-z]", password): score += 1
    if re.search(r"[0-9]", password): score += 1
    if re.search(r"[^A-Za-z0-9]", password): score += 1
    if password.lower() not in common_passwords: score += 1

    entropy = calculate_entropy(password)

    if entropy < 28:
        return "Weak"
    elif entropy < 36:
        return "Medium"
    elif entropy < 60:
        return "Strong"
    else:
        return "Very Strong"

password = input("Enter a password: ")
print("Password Strength:", check_strength(password))
```

```
import re
import math

common_passwords = ["password", "123456", "qwerty", "admin", "letmein"]

def calculate_entropy(password):
    pool = 0
    if re.search(r"[a-z]", password): pool += 26
    if re.search(r"[A-Z]", password): pool += 26
    if re.search(r"[0-9]", password): pool += 10
    if re.search(r"[^A-Za-z0-9]", password): pool += 32
    return len(password) * math.log2(pool) if pool else 0

def check_strength(password):
    score = 0

    if len(password) >= 8: score += 1
    if re.search(r"[A-Z]", password): score += 1
    if re.search(r"[a-z]", password): score += 1
    if re.search(r"[0-9]", password): score += 1
    if re.search(r"[^A-Za-z0-9]", password): score += 1
    if password.lower() not in common_passwords: score += 1

    entropy = calculate_entropy(password)

    if entropy < 28:
        return "Weak"
    elif entropy < 36:
        return "Medium"
    elif entropy < 60:
        return "Strong"
    else:
        return "Very Strong"

password = input("Enter a password: ")
print("Password Strength:", check_strength(password))
```

Screenshots / Results

Weak password result

```
import math

common_passwords = ["password", "123456", "qwerty", "admin", "letmein"]

def calculate_entropy(password):
    pool = 0
    if re.search(r"[a-z]", password): pool += 26
    if re.search(r"[A-Z]", password): pool += 26
    if re.search(r"[0-9]", password): pool += 10
    if re.search(r"^[A-Za-z0-9]", password): pool += 32
    return len(password) * math.log2(pool) if pool else 0

def check_strength(password):
    score = 0

    if len(password) >= 8: score += 1
    if re.search(r"[A-Z]", password): score += 1
    if re.search(r"[a-z]", password): score += 1
    if re.search(r"[0-9]", password): score += 1
    if re.search(r"^[A-Za-z0-9]", password): score += 1
    if password.lower() not in common_passwords: score += 1

    entropy = calculate_entropy(password)

    if entropy < 28:
        return "Weak"
    elif entropy < 36:
        return "Medium"
    elif entropy < 60:
        return "Strong"
    else:
        return "Very Strong"

password = input("Enter a password: ")
print("Password Strength:", check_strength(password))
```

```
Enter a password: 123456
Password Strength: Weak
```

Medium result

```

common_passwords = [ password , 123456 , qwerty , admin , letmein ]

def calculate_entropy(password):
    pool = 0
    if re.search(r"[a-z]", password): pool += 26
    if re.search(r"[A-Z]", password): pool += 26
    if re.search(r"[0-9]", password): pool += 10
    if re.search(r"[^A-Za-z0-9]", password): pool += 32
    return len(password) * math.log2(pool) if pool else 0

def check_strength(password):
    score = 0

    if len(password) >= 8: score += 1
    if re.search(r"[A-Z]", password): score += 1
    if re.search(r"[a-z]", password): score += 1
    if re.search(r"[0-9]", password): score += 1
    if re.search(r"[^A-Za-z0-9]", password): score += 1
    if password.lower() not in common_passwords: score += 1

    entropy = calculate_entropy(password)

    if entropy < 28:
        return "Weak"
    elif entropy < 36:
        return "Medium"
    elif entropy < 60:
        return "Strong"
    else:
        return "Very Strong"

password = input("Enter a password: ")
print("Password Strength:", check_strength(password))

```

Enter a password: pokemon
Password Strength: Medium

Strong/Very Strong result

```
import math

common_passwords = ["password", "123456", "qwerty", "admin", "letmein"]

def calculate_entropy(password):
    pool = 0
    if re.search(r"[a-z]", password): pool += 26
    if re.search(r"[A-Z]", password): pool += 26
    if re.search(r"[0-9]", password): pool += 10
    if re.search(r"[^A-Za-z0-9]", password): pool += 32
    return len(password) * math.log2(pool) if pool else 0

def check_strength(password):
    score = 0

    if len(password) >= 8: score += 1
    if re.search(r"[A-Z]", password): score += 1
    if re.search(r"[a-z]", password): score += 1
    if re.search(r"[0-9]", password): score += 1
    if re.search(r"[^A-Za-z0-9]", password): score += 1
    if password.lower() not in common_passwords: score += 1

    entropy = calculate_entropy(password)

    if entropy < 28:
        return "Weak"
    elif entropy < 36:
        return "Medium"
    elif entropy < 60:
        return "Strong"
    else:
        return "Very Strong"

password = input("Enter a password: ")
print("Password Strength:", check_strength(password))
```

```
Enter a password: MyS3cureP@ss
Password Strength: Very Strong
```

Testing Approach

Test Cases Used

1. Length testing: very short passwords
2. Character variety testing:

Only uppercase

Only numbers

Only symbols

3. Mixed input testing:

Password:

abc123

Abc!123

4. Common password testing:

Password:

123456

5. Complex password testing:

Password:

u!R2@fKbZ

Expected Outputs

Short/simple → Weak

Medium variety → Medium

Complex → Strong/Very Strong

Challenges Faced

1. Designing accurate entropy thresholds
2. Creating balanced scoring rules
3. Avoiding false positives for strong passwords
4. Handling edge-case inputs (unicode, spaces, empty password)

Learnings & Key Takeaways

*Learned importance of password security

*Understood entropy and randomness

*Practiced Python regular expressions

*Gained experience in modular code design

*Understood how to analyze user input safely

Future Enhancements

1. GUI Interface using Tkinter or PyQt
2. Web-based version using Flask
3. Mobile App using Kivy
4. Real-time suggestions to improve password
5. Integration with password managers
6. AI-based password strength prediction

References

1. Python Official Documentation – <https://docs.python.org>
2. NIST Password Guideline
3. OWASP Authentication Guide
4. Online research on entropy and password strength formulas