**AFRICAN INSTITUTE FOR MATHEMATICAL SCIENCES**

**(AIMS RWANDA, KIGALI)**

**Group 2**

**Grace KITONYI** *(Group Leader)*
**Musonda KATAI**
**Latifah AKIMANA**
**Tchandikou OUADJA FARE**

**Project:** Check a Password

**Course:** Python Programming

**Date:** October 8, 2025

# Contents

**Abstract**

This report presents a collaborative group project focused on developing a Python program that evaluates the strength of user passwords based on essential security standards. The system checks for adequate length, character diversity, and the absence of predictable or repetitive patterns, while also rejecting commonly used passwords that are vulnerable to attacks. By integrating multiple validation rules within a clear and modular structure, the group aimed to promote secure password practices through an interactive and educational tool. The program provides immediate, actionable feedback to guide users toward creating stronger passwords, reinforcing both good programming design and cybersecurity awareness.

# 1 Introduction

Password security is a critical aspect of digital safety, as weak or predictable passwords significantly increase the risk of account compromise. In response to this challenge, our group designed a Python program that systematically evaluates password strength using a set of predefined security rules. These include checks for sufficient length, mixed character types, avoidance of repetition, and exclusion of common passwords. The program's interactive feedback mechanism helps users iteratively refine their passwords, fostering awareness of strong authentication practices.

# 2 Requirements

A password is accepted if all rules in Table 1 are satisfied.

Table 1: Validation rules enforced by the checker

| **Category** | **Rule** |
|---|---|
| Character mix | At least one uppercase, one lowercase, one digit, one special char |
| Whitespace | Must contain no spaces |
| All alpha/numeric | Must not be entirely alphabetic or entirely numeric |
| Repetition | No $\geq 3$ identical consecutive characters (e.g., `aaa`, `111`) |
| Common list | Must not match a small list of very common passwords |

# 3 Design Overview

The overall design of the program follows a modular and user-centered approach, separating the core validation logic from the user interaction component. This structure improves readability, reusability, and ease of testing.

1. **Password Evaluation Function (`check_password(password)`)**
   This function is responsible for verifying whether a given password meets all predefined security conditions. It processes the input value, performs a series of checks (length, character categories, repetition, and common-password detection), and stores feedback

messages in a list called `comments`. It returns two outputs:

- `is_good`: a variable with a boolean value indicating whether the password passed all checks with a True value or False if it did not meet all defined rules.
- `comments`: a list containing specific recommendations for improvement.

Following the requirements of the assigned project, the function is designed to be pure, meaning it does not rely on external variables which allows it to be reused or imported into other applications such as web interfaces or authentication systems.

2. **Main Program Loop**
   The main script runs a `while` loop that continuously prompts the user to enter a password until a strong one is provided. After each input, the program calls `check_password(password)` and displays the feedback from `comments`. This interactive process ensures the user understands which requirements are missing and how to strengthen their password before proceeding.

Together, these components create a simple yet effective design: the function handles the logic, and the loop manages user interaction. This separation of concerns makes the code easy to maintain, extend, and debug.

# 4 Method and Implementation Details

## 4.1 Function Signature and Feedback List

Listing 1 shows the entry point of the program, defined by the function `check_password(password)`. Inside the function, a list named `comments` is created to collect feedback messages, and an extensive list of common passwords is initialized for comparison during validation. This setup establishes the foundation for all subsequent password checks.

Listing 1: Function definition and initialization of feedback and common password list.

```
    def check_password(password):
    """Check password
    Input: password
    Output: tuple (is_good, comments) where:
    - is_good: True if all rules are met
    - comments: list of unmet-requirement messages
    """
    comments = [] # Collects feedback about password strength

    common_passwords = [
    "password", "123456", "qwerty", "letmein", "123456789", "12345678",
    "12345", "qwerty123", "111111", "123123", "abc123", "password1",
    "iloveyou", "000000", "monkey", "dragon", "sunshine", "football",
    "admin", "welcome", "login", "princess", "solo", "starwars",
    "baseball", "hello", "freedom", "whatever", "trustno1", "654321",
    "superman", "asdfghjkl", "pokemon", "liverpool", "charlie", "computer",
    "michelle", "jordan", "tigger", "purple", "ginger", "summer", "ashley",
    "buster", "hannah", "michael", "daniel", "hunter", "shadow", "minecraft",
    "qwertyuiop", "qazwsx", "qwert", "qwe123", "qweasd", "1q2w3e4r", "1qaz2wsx",
    "12qwaszx", "q1w2e3r4", "zaq12wsx", "zaq12wsxc", "admin123", "welcome123",
    "password123", "letmein123", "pass123", "secret", "default", "root", "user",
    "guest", "oracle", "mysql", "postgres", "11111111", "aaaaaa", "abc12345",
    "abc123456", "qwerty1", "qwerty12", "qwerty1234", "qwerty!", "1234567",
    "987654321", "696969", "football1", "monkey1", "dragon1", "p@ssw0rd",
    "pa$$w0rd", "passw0rd", "password!", "password01", "welcome1", "admin1",
    "admin01", "login123", "letmein1"
    ]
```

## 4.2 Length Validation

The first guard checks boundary conditions for length. Short passwords are easily brute-forced; excessively long inputs may be impractical for users or systems.

Listing 2: Length check.

```
    if len(password) < 8 or len(password) > 64:
    comments.append("* The password should be at least 8 characters long and at most
        64 characters long.")
```

## 4.3 Character Categories and Special Set

As the program goes through the password one character at a time, it sets each flag to True when the corresponding condition is met (for example, when an uppercase letter, a lowercase letter, a number, or a special character is found). This makes the checking process simple and easy to understand.

Listing 3: Scanning once to set character-category flags.

```
    has_upper = has_lower = has_digit = has_special = has_space = False
    special_characters = "!@#$%^&*()-_+={}[]:;,<.>?/\\|‘"

    for ch in password:
    if ch.isupper():
    has_upper = True
    elif ch.islower():
    has_lower = True
    elif ch.isdigit():
    has_digit = True
    elif ch in special_characters:
    has_special = True
    elif ch.isspace():
    has_space = True
```

We then add targeted feedback for any missing category.

Listing 4: Targeted feedback for missing character classes.

```
    if not has_upper: comments.append("* Add at least one uppercase letter.")
    if not has_lower: comments.append("* Add at least one lowercase letter.")
    if not has_digit: comments.append("* Add at least one number.")
    if not has_special: comments.append("* Add at least one special character.")
    if has_space: comments.append("* The password must not contain any spaces.")
```

## 4.4 Avoiding Passwords Made of Only Letters or Only Numbers

Passwords made up entirely of letters or numbers are easy to guess and therefore insecure. To address this, the project enforces the use of a combination of letters, numbers, and other character types to strengthen password protection.

Listing 5: Alpha-only and digit-only rejections.

```
    if password.isalpha():
    comments.append("* The password must not be entirely alphabetic.")
    if password.isdigit():
    comments.append("* The password must not be entirely numeric.")
```

## 4.5 Consecutive Repetition Constraint

The program also checks for any sequence of three identical characters in a row to prevent users from choosing weak and repetitive patterns.

Listing 6: Reject 3+ identical consecutive characters.

```
    for i in range(len(password) - 2):
    if password[i] == password[i+1] == password[i+2]:
    comments.append("* The password must not contain 3 or more identical consecutive
        characters or numbers.")
    break
```

### 4.6 Rejecting Common Passwords

Earlier, we defined a list of common passwords that are widely used and often exposed in data breaches. In this part of the code, the program checks whether the user's password matches any item in that list. If a match is found, a comment is added to prompt the user to choose a more unique and secure password.

Listing 7: Checking if the password appears in the list of common passwords.

```
    for pwd in common_passwords:
    if password == pwd:
    comments.append("* The password must not be a common password.")
```

### 4.7 User Interaction Loop

The program keeps prompting until all requirements are met, giving specific guidance at each failure.

Listing 8: Main loop: prompt until strong password.

```
    while True:
    password = input("Enter a password: ")
    is_good, observations = check_password(password)

    if is_good:
    print(is_good)
    print("** Great, this is a strong password. **")
    break
    else:
    print(is_good)
    print("Oops your password is weak. Follow these comments to make it strong")
    print("= == == == == == == == == == == == == === =")
    for comment in observations:
    print(comment)
    print("= == == == == == == == == == == == == === =")
    print("Please enter a strong password.")
```

## 5 Algorithm: Password Strength Checker

This section summarizes the logical flow of the password validation program. The algorithm outlines all major steps from initialization to user interaction and is presented below:

1. **Start**
2. **Define the function** check_password(password):
   (a) Create an empty list comments to store feedback messages.
   (b) Create a list common_passwords containing commonly used weak passwords.
   (c) **Check password length:**

- If the password length is less than 8 or greater than 64, add the comment: "Password should be 8-64 characters long."

(d) Initialize five flags: has_upper, has_lower, has_digit, has_special, and has_space, all set to False.

(e) Define a string special_characters containing symbols such as `!@#$%^&*()-_+={}[]:;,<.>?/\'|`

(f) **For each character in the password:**

- If it is uppercase → set has_upper = True.
- Else if lowercase → set has_lower = True.
- Else if a digit → set has_digit = True.
- Else if in special_characters → set has_special = True.
- Else if it is a space → set has_space = True.

(g) **Check for missing requirements:**

- If has_upper is False → add comment: "Add at least one uppercase letter."
- If has_lower is False → add comment: "Add at least one lowercase letter."
- If has_digit is False → add comment: "Add at least one number."
- If has_special is False → add comment: "Add at least one special character."
- If has_space is True → add comment: "Password must not contain spaces."

(h) **Check if password is entirely alphabetic or numeric:**

- If password has only letters → add comment: "Password must not be entirely alphabetic."
- If password has only digits → add comment: "Password must not be entirely numeric."

(i) **Check for three identical consecutive characters:**

- Loop from the first to the third-last character.
- If password[i] == password[i+1] == password[i+2] → add comment: "Password must not contain 3 or more identical consecutive characters." Then stop the loop.

(j) **Check if password is common:**

- If the password matches any item in common_passwords → add comment: "Password must not be a common password."

(k) **Decide password strength:**

- If comments is empty → set is_good = True.
- Else → set is_good = False.

(l) Return the tuple (is_good, comments).

3. **Main Program Loop:**

(a) Repeat:

- Ask the user to "Enter a password:".
- Call check_password(password) and get (is_good, observations).
- If is_good is True → print "Great, this is a strong password." and stop the loop.

- Otherwise →
  - Print "Oops, your password is weak.".
  - Display all comments from observations.
  - Prompt the user to enter another password.

4. **End**

# 6 Results and Examples Tested

The program was tested with a variety of passwords to evaluate its accuracy and ensure that all validation rules worked as intended. When executed in the command-line version, the program provides clear and immediate feedback to the user, indicating whether the entered password is weak or strong. For weak passwords, specific suggestions are displayed to guide the user on how to improve the password by adding missing elements such as uppercase letters, digits, or special characters, or by avoiding repetition and common patterns. Once all conditions are satisfied, the program confirms that the password is strong and terminates successfully.

In addition to the terminal-based implementation, we developed a simple graphical user interface (GUI) using **Streamlit**. This version allows users to interact with the password checker more intuitively by entering their passwords through a web-based form and receiving real-time feedback on password strength.

The Streamlit interface enhances accessibility and demonstrates how the core validation logic can be easily integrated into user-friendly applications. The deployed web application can be accessed through the following link: **https://check-password-group2.streamlit.app/**.

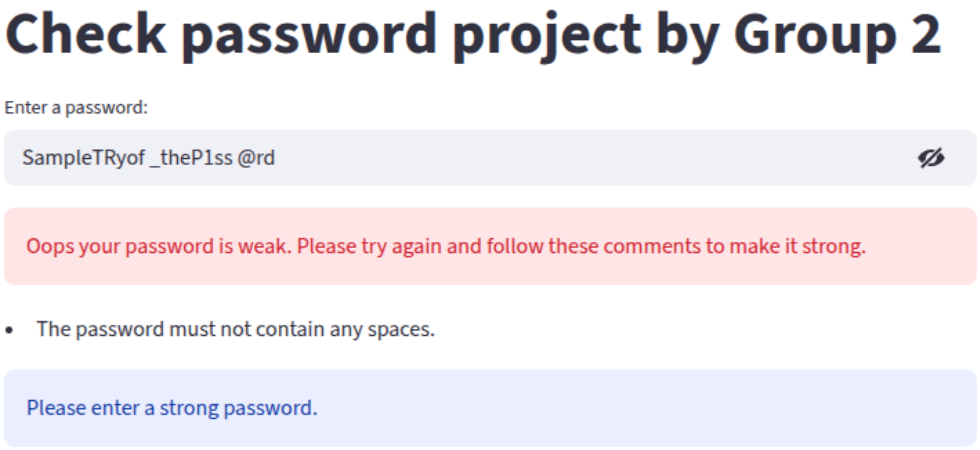


Figure 1: Streamlit UI Password Checker

**The following screenshots illustrate example tested with the function:**

The password checker was tested with multiple input cases to verify that all validation rules

function correctly. Each test demonstrates a specific rule or condition defined in the program. The figures below show the actual command-line outputs obtained for various scenarios.

```
Enter a password:  pass
False
Oops your password is weak. Please try again and follow these comments to make it strong.
= == == == == == == == == == == == == === =
* The password should be at least 8 characters long and at most 64 characters long.
* Please include at least one uppercase letter.
* Please include at least one number.
* Please include at least one special character.
* The password must not be entirely alphabetic.
= == == == == == == == == == == == == === =
Please enter a strong password.
Enter a password:  ↑↓ for history. Search history with c-↑/c-↓
```

Figure 2: Weak password `pass`: Too short, alphabetic only, missing uppercase, digit, and special character.

```
Enter a password:  password
False
Oops your password is weak. Please try again and follow these comments to make it strong.
= == == == == == == == == == == == == === =
* Please include at least one uppercase letter.
* Please include at least one number.
* Please include at least one special character.
* The password must not be entirely alphabetic.
* The password must not be a common password.
= == == == == == == == == == == == == === =
Please enter a strong password.
Enter a password:  ↑↓ for history. Search history with c-↑/c-↓
```

Figure 3: Weak password `password`: Detected as common password and missing character variety.

```
Enter a password:  AbcDef12
False
Oops your password is weak. Please try again and follow these comments to make it strong.
= == == == == == == == == == == == == === =
* Please include at least one special character.
= == == == == == == == == == == == == === =
Please enter a strong password.
Enter a password:  ↑↓ for history. Search history with c-↑/c-↓
```

Figure 4: Weak password `AbcDef12`: Missing a special character.

```
Enter a password:  GoodPass111!
False
Oops your password is weak. Please try again and follow these comments to make it strong.
= == == == == == == == == == == == == === =
* The password must not contain 3 or more identical consecutive characters or numbers.
= == == == == == == == == == == == == === =
Please enter a strong password.
Enter a password:  ↑↓ for history. Search history with c-↑/c-↓
```

Figure 5: Weak password `GoodPass111!`: Contains three identical consecutive digits.

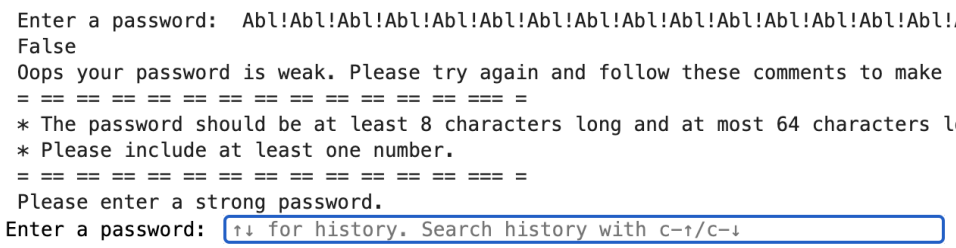


Figure 6: Weak password exceeding length limit (65 characters): Rejected for exceeding the 64-character maximum.

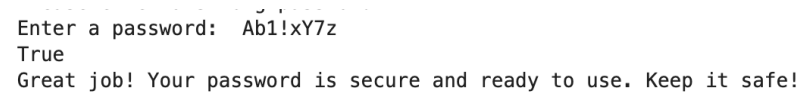


Figure 7: Strong password `Ab1!xY7z`: Meets all conditions and passes validation successfully.

Each test confirms that the program correctly detects weak passwords, provides appropriate feedback messages, and accepts only those meeting all security requirements. These screenshots show how strong the validation is and how the comments are easy for users to understand.

# Conclusion

This project successfully implemented a password strength checker that combines clear logic, efficient execution, and user-friendly feedback. The single-pass scanning approach with explicit flag checks ensures fast performance, while the targeted feedback messages guide users toward creating stronger passwords. The modular structure of the `check_password` function makes the solution highly reusable and adaptable for integration into other systems such as web applications, APIs, or mobile forms.

Furthermore, the addition of a simple Streamlit interface demonstrates how the same core logic can be extended to an interactive, accessible, and visually engaging platform. Overall, the project met its objectives by producing a reliable tool that enforces good password practices and enhances cybersecurity awareness among users.

# Appendix: Full Source code

Listing 9: Full source code.

```
    def check_password(password):
    """
    This function displays true when the password is strong and false when the
        password is not strong with a list of comments or rules that were not met.

    Parameters:
    password

    Returns:
    True (If the password is strong.)
    False (If the password is weak.)
    List of comments (If the password is not strong it returns a list of rules that
        were not met.)
    """
    comments = [] # List to hold comments about password strength
    common_passwords = ["password", "123456", "qwerty", "letmein", "123456789", "
        12345678", "12345", "qwerty", "qwerty123",
    "111111", "123123", "abc123", "password1", "iloveyou", "000000", "letmein",
    "monkey", "dragon", "sunshine", "football", "admin", "welcome", "login",
    "princess", "solo", "starwars", "baseball", "hello", "freedom", "whatever",
    "trustno1", "654321", "superman", "asdfghjkl", "pokemon", "liverpool",
    "charlie", "computer", "michelle", "jordan", "tigger", "purple", "ginger",
    "summer", "ashley", "buster", "hannah", "michael", "daniel", "hunter",
    "shadow", "minecraft", "qwertyuiop", "qazwsx", "qwert", "qwe123", "qweasd",
    "1q2w3e4r", "1qaz2wsx", "12qwaszx", "q1w2e3r4", "zaq12wsx", "zaq12wsxc",
    "admin123", "welcome123", "password123", "letmein123", "pass123", "secret",
    "default", "root", "user", "guest", "oracle", "mysql", "postgres",
    "11111111", "aaaaaa", "abc12345", "abc123456", "qwerty1", "qwerty12", "qwerty1234
        ",
    "qwerty!", "1234567", "987654321", "696969", "football1", "monkey1", "dragon1",
    "p@ssw0rd", "pa$$w0rd", "passw0rd", "password!", "password01", "welcome1",
    "admin1", "admin01", "login123", "letmein1"]

    try:
        if len(password) < 8 or len(password) > 64:
            raise ValueError()
    except ValueError:
        comments.append("* The password should be at least 8 characters long and at
            most 64 characters long.")

    has_upper = False
    has_lower = False
    has_digit = False
    has_special = False
    has_space = False

    special_characters = "!@#$%^&*()-_+={}[]:;,<.>?/\\|‘~"
```

```
43
44      for character in password:
45          if character.isupper():
46              has_upper = True
47          elif character.islower():
48              has_lower = True
49          elif character.isdigit():
50              has_digit = True
51          elif character in special_characters:
52              has_special = True
53          elif character.isspace():
54              has_space = True
55
56      if not has_upper:
57          comments.append("* Please include at least one uppercase letter.")
58      if not has_lower:
59          comments.append("* Please include at least one lowercase letter.")
60      if not has_digit:
61          comments.append("* Please include at least one number.")
62      if not has_special:
63          comments.append("* Please include at least one special character.")
64      if has_space:
65          comments.append("* The password must not contain any spaces.")
66
67      # check if password is entirely alphabetic or numeric
68      if password.isalpha():
69          comments.append("* The password must not be entirely alphabetic.")
70      if password.isdigit():
71          comments.append("* The password must not be entirely numeric.")
72
73      # Check for 3 or more identical consecutive characters or numbers
74      try:
75          for i in range(len(password) - 2):
76              if password[i] == password[i+1] == password[i+2]:
77                  raise ValueError()
78                  break
79      except ValueError:
80                  comments.append("* The password must not contain 3 or more identical
                        consecutive characters or numbers.")
81
82      # Check if password is a common password
83      for pwd in common_passwords:
84          if password == pwd:
85              comments.append("* The password must not be a common password.")
86      is_good = len(comments) == 0 # This will be true when there is no item in the
            comments list
87
88      return is_good, comments
89
90
```

```
91
92  while True:
93      password = input("Enter a password: ")
94      is_good, observations = check_password(password)
95
96      if is_good:
97          print(is_good)
98          print("Great job! Your password is secure and ready to use. Keep it safe!")
99          break
100     else:
101         print(is_good)
102         print("Oops your password is weak. Please try again and follow these comments
                to make it strong.")
103         print("= == == == == == == == == == == == == === =")
104         for comment in observations:
105             print(comment)
106         print("= == == == == == == == == == == == == === =")
107         print("Please enter a strong password.")
```

## Github repository

Check password project