Rapport sur le projet: check pass word

Firstname Middlename Surname (email@aims-senegal.sn)
African Institute for Mathematical Sciences (AIMS), Senegal
Supervised by: Dr Yayé
Institute of Supervisor, Country

26 octobre 2025

Submitted in partial fulfilment of the requirements for the award of Master of Science in Mathematical Sciences at AIMS Senegal

Table des matières

1. Introduction

With the widespread use of digital services and online accounts, password security has become a major concern in the field of **cybersecurity**. Many users still choose weak or predictable passwords, thereby exposing their accounts to brute-force or dictionary attacks. For this reason, it is essential to implement automatic verification mechanisms to assess password strength during their creation.

However, an essential question arises : **how can we ensure that a password chosen by a user is sufficiently secure to withstand unauthorized access attempts ?** Thus, the objective of this project is to design and develop a Python program capable of analyzing a password provided by the user and automatically determining whether it meets a predefined set of security rules.

The program must :

- Check several criteria related to the password's length, composition, and complexity.
- Reject passwords that are too simple, common, or contain repetitive patterns.
- Provide a clear message indicating whether the password is valid, along with the reasons for possible rejection.

To achieve this goal, we will first analyze the **security criteria** of a strong password, including the requirements for length, composition, and complexity. Then, we will implement a program using well-defined functions to verify password validity.

2. Requirements Analysis and Compliance

In this section, we analyze the various requirements established for our project. This analysis allows us to verify whether our Python program properly meets the expected criteria for password security and robustness.

TABLE 2.1 – Code Compliance Analysis with Project Requirements

Project Requirement	**Compliance and Analysis**
Length : Between 8 and 64 characters.	**Compliant.** Checks whether `8 <= len(password) <= 64` and adds an error if not.
Content : At least one uppercase letter, one lowercase letter, one digit, and one special character.	**Compliant.** Uses flags (`has_uppercase`, etc.) to verify each condition and records an error if any are not met. A predefined list of special characters is used.
Spaces : Must not contain spaces.	**Compliant.** Simply checks whether `" " in password`.
Alphabetic/Numeric only : Reject if entirely alphabetic *or* entirely numeric.	**Compliant.** The conditions `is_entirely_alpha` and `is_entirely_numeric` are correctly implemented using character presence flags.
Consecutive characters : Reject if 3 or more identical consecutive characters (e.g., aaa, 111).	**Compliant.** Iterates through the string to detect `password[i] == password[i + 1] == password[i + 2]` and stops at the first occurrence (`break`).
Common passwords : Reject if included in a list of common passwords ("password", "123456", "qwerty", "letmein").	**Compliant.** Uses a list of unsafe passwords and checks whether the user's input (converted to lowercase with `password.lower()`) appears in it.
Function return : Return `True` if valid, `False` otherwise.	**Compliant.** Returns `True` when the error list is empty and `False` otherwise.
Implementation requirements : Pure Python, no external libraries (`re`, `string`).	**Compliant.** The code relies solely on Python's built-in features (loops, strings, lists, conditionals).

3. Adopted Methodologies

3.0.1 Project Workflow

The image below illustrates the overall functioning of the password verification program.

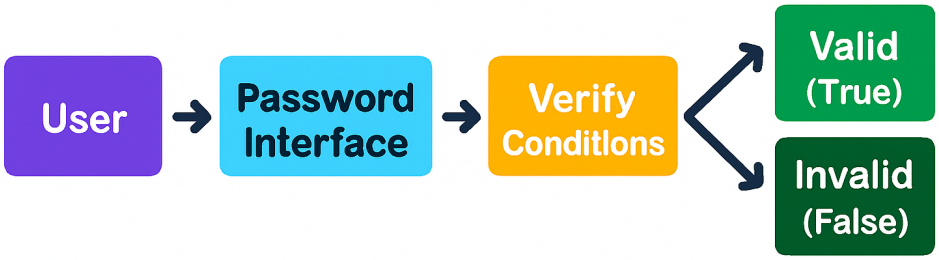

FIGURE 3.1 – Workflow of the Check Password Project

The process proceeds through the following steps :

1. **User :** The user enters a password via the interface.
2. **Password Interface :** The password is passed to the main function `check_password()`.
3. **Verify Conditions :** The program checks the various security rules (length, absence of spaces, presence of special characters, etc.).
4. **Result :**
   - If no errors are detected, the password is **valid** (`True`).
   - Otherwise, it is **invalid** (`False`), and the corresponding error messages are displayed.

3.0.2 Tools Used

To carry out this project, several Python tools and libraries were employed to ensure both the robustness of the validation logic and the simplicity of the user interface :

- **Language :** Python 3

- **Library :** Tkinter (Python's standard library for building intuitive and interactive graphical interfaces.)
- **Code Editor :** **J**upyter **N**otebook
- **Execution Environment :** Python terminal or Tkinter interface

4. Implementations

4.1 Function Definitions

4.1.1 The `length` Function

This function manages the password length and prompts the user to enter a new password if the required **length constraint** is not met. It is **defined** as follows :

```
def length(password):
    errors = []
    if not (8 <= len(password) <= 64):
        errors.append("The password must be between 8 and 64 characters long.")
    return errors
```

FIGURE 4.1 – Implementation of the password length verification function

4.1.2 The `no_space` Function

This function is used to **handle** spaces in the password entered by the user. It is **defined** as follows :

```
def no_spaces(password):
    errors = []
    if " " in password:
        errors.append("The password must not contain any spaces.")
    return errors
```

FIGURE 4.2 – Implementation of the space verification function

4.1.3 The `character_requirements` Function

This function checks for the presence of at least one uppercase letter, one lowercase letter, one digit, and one predefined special character in the password.

It also **verifies** that the password is not entirely alphabetic or entirely numeric. For each unmet requirement, a descriptive error message is added to a list.

Finally, this error list is returned, providing a complete summary of the password's weaknesses. The function is **defined** as follows :

```python
def character_requirements(password):
    errors = []
    special_chars = "!@#$%^&*()-_+=[]{};:’¨,<.>/?\\|‘£°§"

    has_uppercase = False
    has_lowercase = False
    has_digit = False
    has_special = False

    for char in password:
        if 'A' <= char <= 'Z':
            has_uppercase = True
        elif 'a' <= char <= 'z':
            has_lowercase = True
        elif '0' <= char <= '9':
            has_digit = True
        elif char in special_chars:
            has_special = True
```

FIGURE 4.3 – Part 1 — Implementation of the character composition verification function

```python
    if not has_uppercase:
        errors.append("The password must contain at least one uppercase letter.")
    if not has_lowercase:
        errors.append("The password must contain at least one lowercase letter.")
    if not has_digit:
        errors.append("The password must contain at least one digit.")
    if not has_special:
        errors.append(f"The password must contain at least one special character from: {special_chars}"

    is_entirely_alpha = (has_uppercase or has_lowercase) and not has_digit and not has_special
    is_entirely_numeric = has_digit and not has_uppercase and not has_lowercase and not has_special

    if is_entirely_alpha:
        errors.append("The password cannot be entirely alphabetic.")
    if is_entirely_numeric:
        errors.append("The password cannot be entirely numeric.")
    return errors


print(character_requirements("abc123"))
```

FIGURE 4.4 – Part 2 — Implementation of the character composition verification function

4.2 consecutive_characters

This function **checks** whether the password contains three identical consecutive characters. It iterates through the password in sequences of three characters and adds an error to the list if such a sequence is found. It is **defined** as follows :

```python
def consecutive_characters(password):
    errors = []
    if len(password) >= 3:
        for i in range(len(password) - 2):
            if password[i] == password[i + 1] and password[i + 1] == password[i + 2]:
                errors.append("The password contains three identical characters in a row.")
                break
    return errors
```

FIGURE 4.5 – Implementation of the consecutive character verification function

4.3 commonly_used_password

This function **verifies** whether the provided password belongs to a predefined list of common and insecure passwords. If a match is found, an error message is added to the list that is then returned. It is **defined** as follows :

```python
def commonly_used_password(password):
    errors = []
    INSECURE_PASSWORDS = ["password", "123456", "qwerty", "letmein", "password1", "abcd"]
    if password.lower() in INSECURE_PASSWORDS:
        errors.append("The password is too common and not secure.")
    return errors
```

FIGURE 4.6 – Implementation of the common password verification function

4.4 Final Function (check_password)

- This function verifies the password against all predefined rules.
- It returns `True` if the password is valid, and `False` otherwise.
- If invalid, it displays all detected errors in the console.

It is **defined** as follows :

```python
def check_password(password):
    all_errors = []

    all_errors += length(password)
    all_errors += no_spaces(password)
    all_errors += character_requirements(password)
    all_errors += consecutive_characters(password)
    all_errors += commonly_used_password(password)

    if not all_errors:
        print("Password accepted: all rules are satisfied.")
        return True
    else:
        print("Password rejected. The following errors were found:")
        for error in all_errors:
            print(f"    {error}")
        return False

password = input("give a password: ")
while check_password(password) == False:
    print(False)
    password = input("give another password: ")
print(True)
```

FIGURE 4.7 – Implementation of the main check_password function

4.4.1 Validation Tests

In this section, we illustrate how the code behaves depending on the **password** entered by the user, using several test examples.

TABLE 4.1 – Validation Tests

Password Examples	Results	Error or Confirmation Messages
abc123	False	Missing **uppercase letters**, **special characters**, and length below 8.
qwerty	False	The password is too **common** and insecure.

Validation Tests (continued)

Password Examples	**Results**	**Error or Confirmation Messages**
AAA111!!!!	`False`	Contains 3 identical consecutive characters.
Password1!	`True`	All security criteria satisfied.

5. Conclusion

This project made it possible to design a password verification system that combines algorithmic rigor with a simple and user-friendly interface. By using the Python programming language and the Tkinter library, we developed an application capable of analyzing, in real time, the strength of a password according to several security criteria : length, composition, and complexity.

Each rule was implemented as a dedicated function, ensuring modularity and clarity in the code. The graphical interface provides users with an intuitive experience, offering clear messages and visual indicators for each validation criterion.

This work is part of a broader effort to raise awareness about cybersecurity, highlighting the importance of choosing strong passwords to protect personal data and digital access.

In this context, several improvements could be considered to enhance and extend the program's functionality :

- **Integration of a built-in password manager**, allowing users to securely store and **encrypt** validated passwords.
- **Multilingual extension** of the interface to improve international accessibility.
- **Development of a mobile or web version** of the application for broader usability.