**Report on Password Strength Analysis Algorithm**

**1. Introduction**

The purpose of this report is to provide a detailed explanation of the password strength analysis algorithm implemented in Python using regular expressions and a graphical user interface (GUI) with Tkinter. The algorithm is designed to assess the security of user-input passwords based on specific criteria and provide feedback on their strength. Additionally, the program incorporates password hashing using the SHA-256 algorithm to enhance security.

**2. Algorithm Explanation**

The algorithm consists of three main components:

1. **Password Strength Evaluation**
2. **Graphical User Interface (GUI) Implementation**
3. **Password Hashing**

**2.1 Password Strength Evaluation**

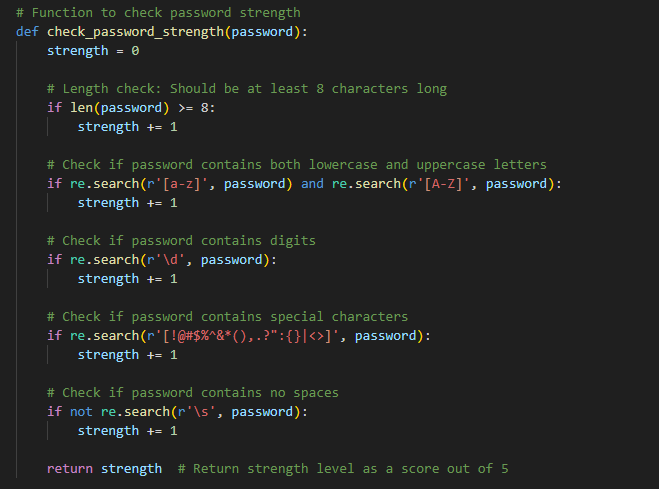
The function check\_password\_strength(password) assesses a password based on five security criteria:

1. **Length Check** - The password must be at least 8 characters long.
2. **Letter Case Check** - It should contain both uppercase and lowercase letters.
3. **Digit Check** - It should include at least one numeric digit.
4. **Special Character Check** - The password must contain at least one special character (e.g., !@#$%^&\*()).
5. **No Space Check** - The password should not contain whitespace characters.

Each criterion met increases the password strength score by 1, leading to a final score ranging from 0 to 5. The scoring system determines the password's security level:

* **5/5** – Very Strong
* **4/5** – Strong
* **3/5** – Medium
* **2/5** – Weak
* **1/5 or 0/5** – Very Weak

This systematic approach ensures a comprehensive evaluation of password security.



**2.2 Graphical User Interface (GUI) Implementation**

The program utilizes Tkinter to create a user-friendly interface for analyzing password strength. The key UI elements include:

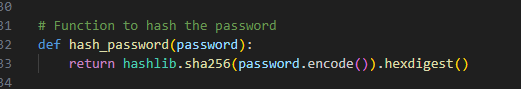
* **Label:** Prompting users to enter a password.
* **Entry Field:** Allows users to input their password securely (masked with \*).
* **Button:** Triggers password strength analysis.
* **Result Label:** Displays the strength of the password.
* **Hashed Password Label:** Shows the hashed value of the password using SHA-256.

The GUI is designed to be simple yet effective, enabling users to quickly check password strength and obtain a secure hashed version.



**2.3 Password Hashing**

The function hash\_password(password) converts the user-input password into a secure, irreversible SHA-256 hash. The hashing process enhances security by ensuring that even if passwords are stored, they remain protected from unauthorized access. The SHA-256 algorithm is widely recognized for its cryptographic strength and resistance to attacks.



**3. Effectiveness of the Algorithm**

The effectiveness of this algorithm is evaluated based on the following aspects:

**3.1 Security Strength**

* The algorithm follows well-established password security principles.
* It discourages weak passwords by enforcing length, complexity, and special character requirements.
* The SHA-256 hashing ensures that even if passwords are stored, they are not in plain text.

**3.2 Usability**

* The Tkinter GUI provides an intuitive interface for non-technical users.
* Users receive immediate feedback on password strength, promoting better password practices. The inclusion of password hashing educates users on secure storage methods.

**3.3 Limitations**

* The algorithm does not enforce password policies; it only provides guidance.
* It does not check against common password lists or dictionary attacks.
* While SHA-256 is strong, it is not the most secure option for password storage; algorithms like bcrypt or Argon2 would be more resistant to brute-force attacks.

**4. Conclusion**

The password strength analysis algorithm is a practical tool for evaluating password security and educating users on best practices. By combining regular expressions, GUI-based feedback, and cryptographic hashing, it effectively enhances password security awareness. While the algorithm is robust, additional features such as checking against breached password databases and implementing more advanced hashing methods could further improve its effectiveness.

Future improvements could include:

* **Integration with breached password databases (e.g., Have I Been Pwned API)**
* **Using stronger password hashing algorithms such as bcrypt or Argon2**
* **Providing password generation suggestions based on best security practices**

Overall, this algorithm serves as a foundational tool for password security analysis and can be expanded for more advanced applications.