Project #2 – Bulk Password Checker

Objectives

- Code a script / program that produces the required output.

Your mission...

You goal is to create a Python script (Program) that will present the user with two different options.

1) Open an **Input text file (Users-Pwds.txt)** that contains usernames and passwords. Then create an **Output text File (Users-Pwds-Chked.txt)**. The output text file will contain the username, password, **and strength (Weak or Moderate or Strong)**.

2) **A Password Generator** that will prompt the user for a username and randomly generate a STONG password. Then it will append the new username and password to the **Input text file (Users-Pwds.txt)**.

Requirements

Create a source code / script file named ***Your_Initials*_PwdChk.py**.
For example, if your name were Meg Griffin, you would name the script **MG_ IPv4BreakDown.py**.

Include a documentation header (as well as comments in your code as well).

Write a short program that meets all of the following requirements.

0. Present the user with the **Title of your program** and **two menu options** (Then select either option 1 or 2).
   0.1. You are to create your own text descriptions for option 1 and 2 to output to the user.
   0.2. Challenge for the stronger programmers (not required).
   The user only needs to press 1 or 2 (no need to press enter) and you cannot use the input function.
1. If they select **Option 1**:
   1.1. Open an **Input text file (Users-Pwds.txt)** stored in the same directory as your Python file.
      1.1.1. The file contains usernames and passwords (Comma separated and provided with the assignment).
      1.1.2. **BONUS**: Use the input function and ask the user for the path to the input file.
         1.1.2.1. If they enter "**C:\folder\file.txt**" you will need to convert to a valid format for python.
         1.1.2.2. If they do not enter the correct **file extention (.txt)** then make the correction as well.
   1.2. Process all the lines in the input file, checking each password string for its strength.
      1.2.1. The password checking must be done in **a function** (Your choice of function name)
      1.2.2. The function must have **only one argument**, a string that contains the password (not username).
      1.2.3. It will then rank the strength of the password. **WEAK / MODERATE / STRONG** (See on Page 2)
      1.2.4. Then **return a string** with the rank.
   1.3. The program will create an **Output text File (Users-Pwds-Chked.txt)**.
      1.3.1. The output text file will contain the username, password, **and strength.**
      Again the file will be comma separated (chad.h,123abc,Weak) and one user per line**.**
   1.4. The program will then tell the user **the number of passwords checked** from the file and the **feedback** (output) can be found **in the file** (provide filename).
   1.5. Then the program will End. (See Step 3.)
      1.5.1. Challenge for the stronger programmers (not required).
      Return to the menu and add Option 3 to exit the program.

NOTE: This is **NOT** a group assignment and must be completed individually.

2. If they select **Option 2**:
   2.1. Prompt the user for a username (No more the 20 characters in length).
   2.2. Create **a Function** that will have no (zero) arguments. (Your choice of function name)
      2.2.1. The Function will randomly generate a STONG password (Meeting the STRONG Requirments).
      2.2.2. If you need help, here is a suggestion (not required if you have a method already).
         2.2.2.1. Generate Random characters the meet the requirements in a string.
         2.2.2.2. Use the shuffle() function from the random module to randomize the order of the characters.
      2.2.3. Then return a string that contains the new generated password.
      2.2.4. **YOU CANNOT USE CODE FROM THE INTERNET FOR THIS or ANY PART of your program.**
   2.3. Then output the new **username** and **password** to the screen**.**
   2.4. Ask the user if they would like this **saved (Y or N).**
      2.4.1. **If Y**: Open the **Input file (Users-Pwds.txt)** and **append** the **username,password** to the EOF.
      2.4.2.**BONUS**: Use the input function and ask the user for the path to the output file.
         2.4.2.1. If they enter "**C:\folder\file.txt"** you will need to convert to a valid format for python.
         2.4.2.2. If they do not enter the correct **file extention (.txt)** then make the correction as well.
      2.4.3. **If N:** Ask if they would like to generate a different password for this user **(Y or N)**.
         2.4.3.1. **If Y:** loop back, were you called your function, and then complete Steps 2.3 / 2.4 (above) again**.**
         2.4.3.2. **IF N:** resume program sequence.
   2.5. Then the program will End. (See Step 3.)
      2.5.1.Challenge for the stronger programmers (not required).
      Return to the menu and add Option 3 to exit the program.
3. When the **program ends** output to the screen: "This program is courtesy of: ***YourName*** "

The following are the requirements for WEAK / MODERATE / STRONG password.

Passwords can contain (not required) any of the following requirements:

i) Lower case letters (a – z).
ii) Upper case letters (A – Z).
iii) Numbers (0 – 9).
iv) Symbols (! + - = ? # % * @ & ^ $ _)

1) A **WEAK** Password is defined as:
   a. Contains **only 1 or 2** from the above 4 items (i – iv).
   b. **Less than 8** characters in length.

2) A **MODERATE** Password is defined as:
   a. Contains **3** from the above 4 items (i – iv)
   b. **Between 8 to 10** characters in length.

3) A **STRONG** Password is defined as:
   a. Meets all **4** of the above items (i – iv)
   b. **Greater than 10** characters in length.

NOTE: If you do not meet all requirements for **STRONG** you then fall to **MODERATE**.
If you do not meet all requirements for **MODERATE** you then fall to **WEAK**.
HINT: Start checking IF STRONG then IF MODERATE then ELSE Weak.

NOTE: This is **NOT** a group assignment and must be completed individually.

TIPS: This might feel a bit over whelming. If so, I recommend the following approach.

Step 1: Create the Function that will be needed for Option 1.

Just call and pass it test passwords (Static Literal Variables) to test the function.

Step 2: Create the Function that will be needed for Option 2.

Call the function and print the output to the screen to see how the passwords are being generated.

You can pass the output of this function (a random password) to the function above to see if it is weak / moderate / strong. The second function should produce a strong password.

Step 3: Create a simple program the reads the sample input file. Breaks each line into Username and Password (Comma Separated) then passes the password only to the function in step 1. For now just output to the screen the feedback for each user-pwd (line) in the test file.

Step 4: Create a simple program that asks for Option 1 or 2.

Step 5: Combine all these steps into one program that meets the above program requirements.

Marking Scheme

Marks	*Description*
2	Code runs correctly (Meets the requirements) without crashing (All or nothing)
2	Has a comment header block listing program name, purpose, coder, and date. Comments are used throughout to explain what is happening (All or nothing)
2	Creating needed variables and sequences (List or Tuple or Dictionary) needed.
2	Throughout the program if the user is prompted for inputs from the keyboard. - Accepts appropriate data types. - Using a try: (assert: or raise:) except: block if entry is invalid. - If user enters an invalid information, ask to try again.
6	A Function that is passed a password and returns if is weak / moderate / strong. - Meeting the above definitions.
4	A Function that, when called, returns a random strong password. - Meeting the above definitions.
4	Properly Opens a File for READ only and loops through all the lines in the file. - Using a try: except: block for an IOError. - Properly opens the file. - Retrieves the names and passwords from the file. - Breaks the Username(Comma)Password lines apart for processing.
4	Properly Opens a File for WRITE and outputs the feedback in the file. - Using a try: except: block for an IOError. - Properly opens the file. - Merges the Username(Comma)Password(Comma)Strength for the file. - Stored the Usernames, passwords, strength (1 per line) to the file.
2	Output shows the correct answers.
2	Screen input / output is in a nice format. (Your Design)
3 (B)	**BONUS**: Ask the user for the path to the input/output files. IMPORTANT: Main requirements must be meet to earn any bonus marks.
30	**TOTAL Marks** (Worth 20% of your final grade in INFO-1249)

NOTE: This is **NOT** a group assignment and must be completed individually.