CPSC 1301, Computer Science I Programming Exercise 10

Week 10, Passwords.py

1 Introduction

This activity consists of five programming exercises. The following exercises are open book and open note. You are free to use any written documentation you wish. However, these are individual exercises, and you cannot consult with each other in writing your programs.

This programming exercise has five parts consisting of five requirements. The grade for each requirement is indicated, for a maximum of 100 points. At a minimum, your program must compile successfully and run.

2 Exercise requirements

You have been assigned the implementation of a password system that runs in memory. Write a console program that does two things: (1) it collects a plain text password from an individual user and stores the password in a string cipher, and (2) it verifies a user by username and password. Your user interface should have five options: (1) save a new password for a specific username, (2) authenticate a specific username/-password pair, (3) change a user's password, (4) display all users, and (5) exit the application. On exit, all username/password pairs will be lost

Passwords entered at the console prompt are to be in plain text. All password saved in memory are to be encoded. Imagine that someone else is writing a database module to afford persistent storage of usernamne/password pairs. You do not want anyone with access to the database to be able to read the plain text of passwords.

You have three challenges in this exercise. First, you need to construct a user interface that presents the user with the three options listed above. Second, you need to discover how you can convert strings to unintelligible collections of random characters — this is called *hashing*. Check the documentation — it's your friend. Third, you need to define a data structure (collection) that can insert values and search values very rapidly. *In this programming exercise, you are required to use a* ***generic dictionary****.* Make sure you import `hashlib` in your header.

As always, work the problem out by hand first, then implement your natural language solution in Python code. The exercise has some challenges, but it is decidedly on the easy side. Solve the problem first in English, then rewrite it in the language that the machine understands.

`interface()` This function will display a user interface, prompt the user to make a selection, accept the user's input, and return the input to the main loop. This is the variable `what` in the template below. Perhaps the user interface might look something like this:

```
---------------------------------------------------------------------

            PASSWORD AUTHENTICATION SYSTEM

            Please select one option:

            1. Establish an account
            2. Authenticate a user
            3. Chaange a password
            4. Display all users
```

```
            9. Exit

            Enter selection:

-----------------------------------------------------------------
```

`add_new_user()` This function will prompt the user to enter a new username and password. It will hash the password and add the password to the application database. There is no need to check to see if the user already exists, but you can implement this if you wish.

`authenticate_user()` This function will prompt the user to enter a username and password. If the username entered by the user is not in the application database, the applicatin will inform the user and return tothe main menu. If the password does not match, the application will inform te user and return to the main menu. If both the username and password match,the application will inform that the user has been authenticated.

`change_password()` This function will prompt the user to enter a username and password. If both the username and the password match, the application will prompt the user for a new password, hash the new password, and add the new hashed password to the database.

`display_all_users()` This function erely disolays all users and the (hashed) passwords.

`get_hass(PWD)` This is simply a convenience function that accepts a plain text password and returns the hashed password.

3 Sample output

See the demonstration in class for sample output.

4 Starter template

```
1  #!python
2  # Name: Passwords.py
3  # Author: Your Name
4  # Date: current date
5  # Purpose: password hashing exercise
6
7  from os import system
8  import hashlib
9
10 def hello():
11     system("cls")
12     print("Hello from 'Passwords.py'")
13
14 def interface():
15     pass
16
17 def add_new_user():
18     pass
19
20 def authenticate_user():
21     pass
22
23 def display_all_users():
24     pass
25
```

```
26 def change_password():
27     pass
28
29 def get_hash(password):
30     pass
31
32 #main function executes the defined functions
33 if __name__ == '__main__':
34     hello()
35     users = {}
36     while True:
37         what = interface()
38         if what == 1:
39             add_new_user()
40         elif what == 2:
41             authenticate_user()
42         elif what == 3:
43             change_password()
44         elif what == 4:
45             display_all_users()
46         elif what == 9:
47             system("cls")
48             break
49         else:
50             system("cls")
51             print("Sorry, but I do not understand")
52
53     print("Goodbye")
```