Project 3 (50 points + 5 points Bonus): Data Structures

Task 1: Password Management System (25 pts)

The following describes the required logistics and basic requirements. However, feel free to add your own design features.

main() function:

1. Create two lists (you choose how to enter/initialize these two lists) usernames and passwords, e.g. usernames = [‘lyang’, ‘kSimon’, ‘danny’, …], passwords = [‘sheCodes#123’, ‘catchAllGood1%’, ‘@my2Choices’, …]. The usernames and passwords must meet the following requirements: Usernames must contain letters (lowercase as well as uppercases can be used). Usernames are unique (i.e. no two usernames can be the same.) Passwords must meet the following requirements: (1) at least 8 characters, (2) contains at least one lowercase letter, one uppercase letter, one digit, and one special character (non-letter or digit). We assume the usernames and passwords in the initialized lists have met the requirements (your responsibility to ensure that.)
2. Zip two lists to form a dictionary of usernames/passwords with the username as the key.
3. Call functions (see below) to perform: (1) login users; acknowledge if login successful or not. (2) change passwords; acknowledge if password change successful or not. (3) create new user; if successful, welcome the new user. If not, acknowledge it.

The following functions are needed:

1. A function to authenticate/login a user, i.e. it takes the dictionary as a parameter, prompts for a username, if username not exist, issue error message, but do allow 3 attempts before the function returns False (unsuccessful). If the username is found, asks for a password. Login the user if password matches, allow 3 tries. If successful, the function returns a Boolean value True.
2. A function to create a new user. Again, the function should take the dictionary as parameter, prompts for a username (verify the username meets the above requirements, i.e. in valid format and not duplicate with existing one) and then a password (verify that the password meets the above requirements). If one of them not valid (okay for one attempt only, but you may design multiple attempts if you wish – optional) the function returns a None value, if both the username and passwords are valid, the function returns the username (so that in main() function, you could acknowledge, welcome ….).
3. A function to update the password. Again, it takes the dictionary as the parameter. Its asks for the username and password of the existing user, login first. If login successful, asks for new password, verify that the password meets the requirement, and updates it, then the function terminates by returning a True value. If any issue (i.e. can’t login or new password doesn’t meet the requirements etc.) the function terminates by returning a False value without updating the dictionary.

Required test run: one test run that covers all (primary) features described above. Initialize the username and password lists as follows, then try to login a user, add a new user, and update password. In each action (login/add/update) try both scenarios (i.e. successful or unsuccessful).

usernames = [‘lyang’, ‘kSimon’, ‘danny’, ‘tomatcpp’, ‘csDept’, ‘CoScpp’, ‘broncoWins’, ‘ponyExp’, ‘BldgAndRooms’, ‘helloKitty’] #note: you may use different usernames, but make sure all are valid.

passwords = [‘sheCodes#123’, ‘catchAllGood1%’, ‘@my2Choices’, ‘123abc;;;’, ‘Hello2Monday$’, ‘GoodFriday@Cpp2’, ‘CS2520@Python’, ‘JavaIsHot2!’, ‘2ManyRainingDays!’, ‘1Startup@Starbucks’]
#note: you may use different passwords, but make sure all are valid.

Task 2: Word Processing (25 points + 5 bonus points)

Two articles are stored in two text files, Python.txt and MachineLearning.txt respectively. The following features should be implemented:

1. Read in the two text files one by one. Prompt the user for the file name, then open the file. If file not found, raise an appropriate exception (not the catch-all exception) and ask user to re-enter.
2. For each text file, read in line by line, split each line into a list. Remove all non-words or punctuations (assume proper words only contain letters) and convert all uppercases to lowercases. Combine all lines/lists into one big list (one list for each file.) Then, display how many lines total and how many words total in each file.
3. Create a set of words (distinct words) from each list, i.e. one set for each file.
4. Use ***set operations*** (one set operation for each subtask) to calculate: (1) total words used in two articles (don’t double count); (2) words appear in article 1 but not in article 2; (3) Words appear in article 2 but not in article 1; (4) words appear in both articles; (5) words not repeated in both articles, i.e. words appearing in one of the articles only. For each subtask, display how many words in each case (i.e. how many elements in each result set), as well as the set content. For example, for question (1), you answer should be in the form of: 105 distinct words used in two articles, and they are: python, machine, learning, …
5. Count word frequency – form a dictionary with words as key and counts (the appearance in both files) as values, display the words and counts with the most frequent words first, e.g. (‘python’, 5), (‘machine’, 4), (‘learning’, 4), … If two words of same count, doesn’t matter which word appears first. Note: if ‘python’ appears in article 1 three times and in article 2 two times, the count for ‘python’ is 5.
6. (Bonus, 5 pts) Draw a word cloud (see sample below) based on the word count frequency.

Required test runs: (1) two attached articles attached. (2) two additional articles of your choice with each article at least 20 lines long.

Submission requirement: Place code and result for each task into .pdf file. Submit pdf file (one pdf only) as well as .py file(s) (may have one or more .py files).

Grading criteria:

Task 1 and Task 2: correctness and proper testing. Bonus: quality of word map/cloud.