Assignment 5

Use the following naming scheme for your program files: `a`*assignment#*`p`*problem#yourname#*`.py` . So your program for problem 1 on this assignment will be named `a5p1bob.py` and your for problem 2 will be named `a5p2bob.py` (adjust to be your name). Please submit all of your .py files to the Moodle dropbox.

Problems

1. ### Tic-Tac-Toe

   Write a program that looks at a list of lists representing a Tic-Tac-Toe game, determines if someone has won, and if so says who. (You can assume the game state has arisen in the course of a legal game).

   Embed your code in the file a5p1kate.py available in the assignment 5 tab of the Moodle.

2. ### Dedup

   Write a program that removes duplicate values from a list, i.e. if the list starts out as `[4, 2, 5, 2, 4]`, then after your code fragment runs it should be `[4, 2, 5]`.

   Embed your code in the file a5p2kate.py available in the assignment 5 tab of the Moodle.

   (A real-life application would be to remove duplicates from a list of IP addresses so you have a list of unique visitors to your site. In this case the list might look like ['10.9.0.31', '199.247.52.3', '10.9.0.31', '43.98.12.4', '72.1.3.55', '199.247.52.3', ...]).

3. Poker Full House

Write a program that decides if a list of card numbers is a full house. A full house is a hand in which 3 cards share one face value and the other two share a different face value, e.g. three sevens and two aces. For example the list [6, 32, 39, 0, 45] is a full house because cards 6, 32 and 45 are sevens and cards 39 and 0 are aces. (See Wikipedia for more examples).

Use the first representation shown in the Moodle notes (Jan. 30-Feb. 3: Aggregate Data Types 1: Lists and Strings), i.e. the one in Representing Playing Cards.

Do not remove your test values from the program so I can see how thoroughly you tested your code. Feel free to adapt one of the test frameworks from the problems above to help you in testing your code.

4. Password checker

Most computer systems require the user to select a password. Unfortunately many users, perhaps most, choose poor passwords, i.e. passwords that are easily guessed, or that can be looked up in a dictionary. Weak passwords present security problems because they compromise the integrity of the system. To avoid weak passwords many systems now test the password the user types, and decide whether to accept it or not. Here is an (abbreviated) excerpt from a manual describing the process on one actual system (where the name of the password program is `passwd`). Think of this as the specification for your program and try to implement it as precisely as possible (if you detect ambiguities in it be sure to ask for clarification).

```
When used to change a password, passwd prompts
everyone for their  old  password, if any. It
then prompts for the new password twice. If the
two copies are not identical the cycle of
prompting for the new password is repeated for at
most two more times. When the two copies are
identical a check is made to ensure that the new
password meets construction requirements.

Passwords must be constructed to meet the
following requirements:
*  Each password must have  at  least  six
characters.
*  Each password must contain at least  one
lower-case alphabetic character, one upper-case
alphabetic character, and at least one numeric or
special character.  In this case,  "alphabetic"
refers  to  all upper or lower case letters.
(These requirements ensure they do not use a
"word", i.e. a string that could be looked up in
a dictionary, or their phone/SIN/etc. number --
two common choices).
*  Each password must differ from the user's
login name and the reverse of that login name.
For this comparison, an upper case letter and its
corresponding lower case letter are equivalent.
*  New passwords must differ from the old in  at
least three  positions.  For this comparison, an
upper case letter and its corresponding lower
case  letter are equivalent.
```

Write a program that behaves as specified by this excerpt.

Notes:

- The real `passwd` program, after validating the proposed new password, encrypts it and writes it to a file, but we're omitting that final step ... for now!
- In actual use the program would make a "system call" to get the current user's login name. Since we have not seen how to do this yet, your program will have to read in the login name.

Here are a couple of sample outputs generated by running a solution program:

```
=============================================
Password Checker
---------------------------------------------
Enter the login name: ttopper
Enter your current password: sp33dKills
Enter new password: duckie
Confirm new password: duvkie
New passwords do not match!
Try again.
Enter new password: duckie
Confirm new password: duckie
Password must contain at least one upper case
letter.
Password not acceptable.
Password not changed.
Done.
=============================================
Password Checker
---------------------------------------------
Enter the login name: ttopper
Enter your current password: sp33dKills
Enter new password: d1cT10nary
Confirm new password: d1cT10nary
Password changed.
Done.
```

Not the friendliest exchanges in the world, but hopefully clear enough.