CS116 Winter 2011 Assignment 10
Due: Monday, April 4 at 4:00pm (Note change of day and time)

The solutions you submit must be entirely your own work. Do not look up either full or partial solutions on the Internet or in printed sources.

Important Information:

1. Read the course Style Guide for Python (Appendix C in the course notes) for information on assignment policies and how to organize and submit your work. In particular, your solutions should be placed in files a10qY.py, where Y is a value from 1 to 4.
2. Be sure to download the interface file to get started.
3. You may use helper functions in your solutions as needed. Simply include them in the same file with your solution, but do not declare them locally. You should follow the full design recipe for your helper functions (which includes testing).
4. If you use `print` statements for debugging purposes, be sure to remove them (or comment them out) before you submit your solutions.
5. Do not copy the purpose directly from the assignment description – it should be in your own words.
6. You may assume that all consumed data will satisfy any stated assumptions, unless indicated otherwise.

Language level: Any Python in the range 2.5.0 to 2.7.1. (Do **not** use Python 3.0 or higher)
Coverage: Module 9 - Module 10

Note: The subject of this assignment (the card game Tichu) is a real game, as described on BoardGameGeek.com and Wikipedia.org. However, the game has been significantly simplified for our context. In cases of conflict between the real game and the version described here, the assignment version will be the ultimate authority.

Tichu is a card game, in which four players compete to attain the highest score. The cards used in the game are much like a common deck of cards: there are four suits (named Jade, Star, Sword, Pagoda) of 13 cards, valued, in increasing order, 2,3,4,5,6,7,8,9,10, Jack, Queen, King, Ace. In each round of the game, players receive 13 cards, and attempt to get rid of their cards by playing increasing "sets" of cards. When it is a player's turn, they may pass (play no cards) or play a set of the cards from their hand, according to the rules of the game. Play continues until no one can play any cards. At that point, the player who last played (and who therefore played the highest "set" of cards) wins all the cards just played, and they are added to the player's taken pile (which is distinct from the hand). That player then gets to lead with another "set" of cards from their hand. The round ends when three of the players have no cards left in their hand. Points are then awarded for that round, based only on the number of fives, tens, and Kings in the set of cards taken by each player.

Each question on this assignment will deal with a small aspect of the game. The following two classes are needed throughout the assignment.

```
# A Tichu_Card has two fields:
# value is either an integer (one of 2,3,4,5,6,7,8,9,10), or
#         a string (one of 'Jack','Queen','King','Ace')
# suit is one of 'Jade', 'Sword', 'Pagoda', 'Star'
class Tichu_Card:
    'Fields: value, suit'

# A Tichu_Player is an object with three fields:
# name is a string for the player's name
# hand is a (listof Tichu_Card), a collection of cards currently
#        in the player's hand (these are the cards the player
#        can choose to play in the round)
# taken is a (listof Tichu_Card), a collection of cards that the
#         player has taken in this round (they can no longer be
#         played in this round).
class Tichu_Player:
    'Fields: name, hand, taken'
```

Note that the `Tichu_Card` and `Tichu_Player` classes include several useful functions:

- `__init__` to assist with initializing objects (you will not call this directly, but it is automatically used when the contructor is called – see the interface for details)
- `__repr__` that produces a string representation of an object (you will not call this directly, but it is automatically used when printing objects of these types)
- `__eq__` that consumes two objects and produces `True` if their field values are equal and `False` otherwise (you will not call this directly, but it is automatically used when you use == to compare two objects of the same type or two lists of objects)

1. Complete a Python function `score` that consumes a `Tichu_Player`, and produces the number of points in that player's taken pile. Points are only awarded for cards with value: 5, 10, and 'King', regardless of suit. Those cards are worth, respectively, 5, 10, and 10 points. For example, if player `p`'s taken pile contains 4 Jade, 10 Sword, 8 Star, 10 Star, King Pagoda, Queen Sword, 5 Pagoda, then `score(p) => 35`.

2. Complete a Python function `load_player` that consumes a string for the name of a file in the current directory which includes information about a player. The file has the following format:
    - The first line is the player's name (you may assume no additional spaces, and that there will always be a newline at the end of this line);
    - Each following line corresponds to a `Tichu_Card`, and has three pieces of information:
        - The string `'h'` or `'t'`, indicating whether the card is in the player's hand (`'h'`) or in the player's taken pile (`'t'`);
        - The card value; and

- The card suit.
- Note that each piece of information is separated by at least one space (though possibly more).

The function produces a `Tichu_Player` with the name in the first line of the file, and whose hand and taken pile correspond to the card information as given in the rest of the file.

For example, suppose the file `sample.txt` has the following lines:

```
Ellen
h 5 Jade
t 8 Sword
t 8 Pagoda
h King Star
```

then calling `load_player("sample.txt")` will create a `Tichu_Player p`, with the name "`Ellen`", whose hand contains 5 Jade and King Star, and whose taken pile contains 8 Sword and 8 Pagoda. Note that the card values will be read in as strings. Be sure to convert numerical values to integers, rather than storing them as strings, for the objects you create.

You may assume that the file, if it exists, contains exactly the type of information described, and the card values and suits are all valid. However, do not assume any order on the cards, and do not assume that the hand or taken pile contain any cards (i.e. a file may just contain just a single line with player's name, in which case the two list fields are set to `[]`). Remember to close the file when you are done!

If the file does not exist, the function prints the string "`Error Opening File`" and returns immediately (producing `None`). Therefore, your solution will need to use a `try-except` block. However, we strongly advise that you complete the file reading component of the question prior to working with the `try-except` block.

To test this function, create several different files with different card lists (including some which are empty) corresponding to different players. Your test cases should check whether the player corresponding to the file information matches the player you expected to be created. You may use == to test for equality of two `Tichu_Player` objects, since the method `__eq__` is provided as part of the `Tichu_Player` and `Tichu_Card` classes.

3. Complete the Python function `save_player` that consumes a `Tichu_Player` and a string for a file name, and prints information about the player to the file with the specified name. The file created by this function should contain the following information:
   - Name of player (followed by newline)
   - For each card c in the player's hand, write a separate line of the form:
     - `h c.value c.suit` (followed by newline)
   - For each card c in the player's taken pile, write a separate line of the form:
     - `t c.value c.suit` (followed by a newline)

Do not make any assumptions about the number of cards the player has. Note that the created file is in the same format as the files used in Question 2 (except that here, all the cards in the hand appear before the cards from the taken pile). Remember to close your file!

For example, consider calling `save_player` for the player created in question 2. In that case, the created file should contain:

```
Ellen
h 5 Jade
h King Star
t 8 Sword
t 8 Pagoda
```

To test this function, consider several different cases for a `Tichu_Player`, and check that the function creates a file with the expected content and format. When writing up your test case, just explain what the file should look like (since you can't actually test for equality within the WingIDE).

4. In this question, you will complete the Python function `is_valid_sequence`, which consumes a string which is the name of a file, and outputs a `Boolean` value, indicating whether or not the collection of moves recorded in the file corresponds to a valid sequence of Tichu plays.

Overview

`is_valid_sequence` will open the consumed file for reading. Each line of the input file contains a representation of a Tichu "set" – a non-empty collection of cards. The order of the file corresponds to the order that the sets of cards were played. The first line in the file corresponds to the first card set played in the sequence – it is called the "lead". The next set of cards played must be "playable" on the lead, the third set must be playable on the second set, etc. If the lead set was a valid Tichu set, and each subsequent set was playable on the previous set, then the file corresponds to a valid round of the game and `True` should be returned. Otherwise, `False` must be returned.

File Format

Each line of the input file will have the following format:
"`val suit, val suit, ..., val suit\n`"
where each `(val, suit)` pair is separated from the next by a comma. The value and suit fields are separated from each other by at least one space. In order to fit the set on a single line, some abbreviations are used: for values, '`J`' means '`Jack`', 'Q' means 'Queen', 'K' means 'King', and 'A' means 'Ace'; for suits, 'J' means 'Jade', 'P' means 'Pagoda', 'St' means 'Star', 'Sw' means 'Sword'. You can assume that the cards entered are always valid cards.

Consider a file "`invalid-seq1.txt`" containing:

```
A J, A P, A St, A Sw
```

```
4 J, 4 St, 4 Sw
2 J, 3 J, 4 J, 5 J, 6 J
```
This corresponds to the following sets:
Ace Jade, Ace Pagoda, Ace Star, Ace Sword
4 Jade, 4 Star, 4 Sword
2 Jade, 3 Jade, 4 Jade, 5 Jade, 6 Jade
respectively, which, as we will see is not a valid set of plays in our Tichu.

Valid Leads
The following types of sets are considered valid leads in our Tichu:

- a single card (e.g. 5 Pagoda or King Jade);
- exactly two cards in which the two cards have the same value field (e.g. 4 Pagoda and 4 Sword are a pair, as are Ace Star and Ace Sword);
- exactly three cards in which the three cards have the same value field (e.g. King Pagoda, King Sword, King Jade);
- exactly four cards in which all the cards have the same value field (e.g. 2 Star, 2 Jade, 2 Sword, 2 Pagoda). This type of set is called a *four-of-a-kind bomb;*
- a set of at least 5 cards, all the same suit, in which the set of cards, when sorted, are numbered consecutively (e.g. 2 Star, 3 Star, 4 Star, 5 Star, 6 Star, as well as 8 Jade, 9 Jade, 10 Jade, Jack Jade, Queen Jade, King Jade). Remember that 'Ace' is the highest card. This type of set is also called a *straight-flush bomb.*

Valid Plays
Generally in Tichu, a set of cards can only be played on a set of the same size and same type (i.e. singles on singles, pairs on pairs, triples on triples), with the values of the card in the sets strictly increasing (e.g. a pair of fives can be followed by another pair, with value 6,7,8,9,10,Jack, Queen, King or Ace). However, *bombs* can be played at almost any time. The following rules are used with bombs:

- a *four-of-a-kind bomb* can be played on a single, pair, or a triple, regardless of the values of the cards;
- a *four-of-a-kind* bomb can be played on another *four-of-a-kind* bomb, as long as the card value of the new set is greater than the card value of the previous set;
- a *four-of-a-kind bomb* cannot be played on a *straight-flush bomb*;
- a *straight-flush bomb* can be played on a single, pair, or a triple, regardless of the values of the cards;
- a *straight-flush bomb* cannot be played on a *four-of-a-kind* bomb;
- a *straight-flush bomb* can be played on another *straight-flush*, as long as the new set contains at least as many cards as the previous set, **and** the lowest card in the new set is larger than the lowest card in the previous set. For example, 3,4,5,6,7 of Jade can be played on 2,3,4,5,6 of Sword, and 8,9,10,Jack, Queen, King, Ace of Star can be played on 6,7,8,9,10 Pagoda. However, 3,4,5,6,7,8,9,10 Sword cannot be played on 3,4,5,6,7,8 Jade (because its lowest card is not greater than the lowest card of the previous set), but 4,5,6,7,8,9 Sword could be played on it.

Example:

Calling `is_valid_sequence` with the file "`valid-seq1.txt`", which contains

```
4 J, 4 St, 4 Sw
8 J, 8 P, 8 Sw
10 J, 10 P, 10 St
K P, K Sw, K St
Q J, Q P, Q St, Q Sw
```

will produce `True`, as triple 4 is a valid lead, triple 8 can be played on triple 4, triple 10 can be played on triple 8, triple King can be played on triple 10, and, finally, the four-of-a-kind Queen bomb can be played on the triple King.