CSE231 **Fall 2016**

Project 10: Thumb and Pouch Solitaire

Overview:

This project is worth 55 points (5.5% of the courses grade). It uses classes and is due **Tuesday**, November 29 at 11:59 PM. Note that this project is **long**.

This game is a variant of "Klondike" (http://politaire.com/help/klondike) solitaire, but easier. Klondike was probably the best known version of solitaire world-wide. For many people, Klondike, with its triangular tableau and alternating colors, is the canonical game of solitaire. It dates back to at least 1912 when it was included in the Mary Whitmore Jones collection with the name "Small Triangle". Microsoft has added to its popularity by including a version of Klondike called "Windows Solitaire" in every version of its operating system since Windows 3.0. As said, the "Thumb and Pouch" solitaire is an easier game than Klondike, as because the cards can be played on tableau cards of any different suit.

First Thing First:

The best way to understand this game is to play it. You can play this game here http://politaire.com/thumbandpouch. We suggest you to play this game couple of times, so that you can plan your program effectively. The interesting thing about this game is you can win almost every time if you play it carefully.

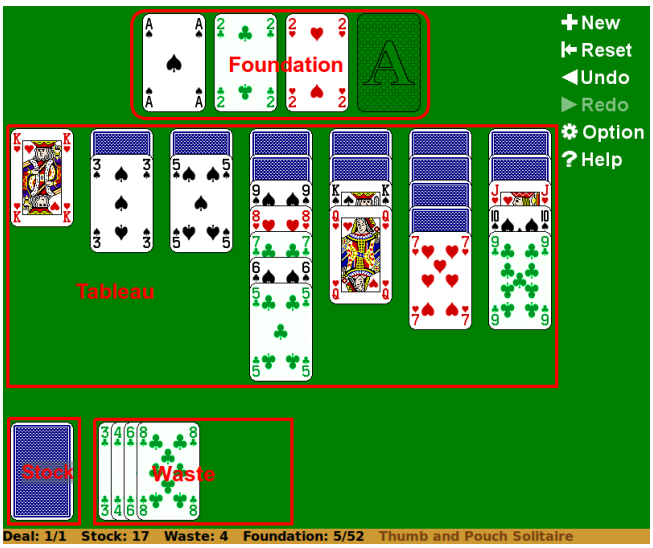
Deliverables:

The deliverable for this assignment is the following file:

`proj10.py` – the source code for your Python program

Be sure to use the specified file name and to submit it for grading via the handin system before the project deadline.

Game Terms: First, let's look at the figure below –



All the top 4 piles are called **"Foundations"**, the seven vertical piles are called as **"Tableau"**, the piles on the bottom-left that are faced down are called **"Stock"** and the piles on the right (faced-up) are called **"Waste"**. The goal of the game is to arrange all the cards into Foundations such that they are sorted according to 4 different suits and ranked from Ace to King.

Game Rules:

- ***Cards:*** One standard deck of 52 cards.
- ***Goal:*** Move all cards to the foundation.
- ***Foundation:*** Four foundation piles. Any ace may be moved to any empty pile in the foundation. A card may be added onto a foundation pile if it is one higher than the old top card of the pile and of the same suit. Thus, the only card that could be played on a Q♣ would be a K♣. Once on the foundation, cards cannot be moved back off.
- ***Tableau:*** Seven tableau piles with one card in the first pile, two cards in the second, three in the third, and so on. Therefore, there will be 28 cards on the tableau. The top card in each pile is dealt face up, all others are face down. The last card on each column is the top card. A card may be added onto a tableau pile if it is one lower than the old top card of the pile and of a different suit. Thus, the cards that could be played on J♦ would be 10♠, 10♥ or 10♣. Cards on the tableau that are not under another card are available for play onto the foundation or any other tableau pile. Empty spaces in the tableau may be filled by any card. Groups of cards in sequence down may be moved from one tableau column to another if no two consecutive cards are of the same suit.
- ***Stock and Waste:*** The remaining cards (i.e. 24 cards) form the stock and there is one waste pile. Each time you draw a card from the stock, the card will be placed on to the waste. Only one pass through the Stock is permitted. Only the top card of the waste is available for play to the tableau or the foundation.

The Detailed Program Specifications:

You will develop a program that allows the user to play "Thumb and Pouch" according to the rules that we have discussed above. The program will use the instructor-supplied **`cards.py`** module to model the cards and deck of cards. To help clarify the specifications, we provide a sample interaction with a program satisfying the specifications in the "Sample Output" section. We provide a program **`proj10.py`** that contains stubs of the required function as well as a driver. The provided program runs as is, but doesn't do anything useful. You are encouraged to create additional functions as needed. Your program must use the import statement for **`cards.py`** — you are not allowed to copy that file into your program. (Your program will be run with our copy of **`cards.py`**).

Program commands: The program will recognize the following commands, the commands can also be entered as a mix of upper/lower case characters (like **`TF`** or **`tf`** or **`tF`** etc.) –

- **`tf x y:`** Moves 1 card from Tableau column **`x`** to Foundation **`y`**.
- **`tt x y n:`** Moves a pile of length **`n >= 1`** from Tableau column **`x`** to Tableau **`y`**.
- **`wf x:`** Moves the top card from the Waste to Foundation **`x`** .
- **`wt x:`** Moves the top card from the Waste to Tableau column **`x.`**
- **`sw:`** Draws one card from Stock to Waste.
- **`r:`** Restarts the game with a re-shuffle, resets everything.
- **`h:`** Displays this menu of choices.
- **`q:`** Quits the game.

Here, **`x`** and **`y`** denote column numbers; Tableau columns are numbered from 1 to 7 and Foundation piles are numbered from 1 to 4.

The program will repeatedly display the current state of the game and prompt the user to enter a command until the user wins the game or enters **`q`**, whichever comes first.

The program will detect report and recover from invalid commands. None of the data structures representing the stock, tableau, or foundation will be altered by an invalid command.

Function Descriptions: We have already supplied stubs for couple of functions, but your implementation can include more functions if needed.

1. The program will use the following function to initialize a game:

   `setup_game() → (foundation, tableau, stock, waste)`

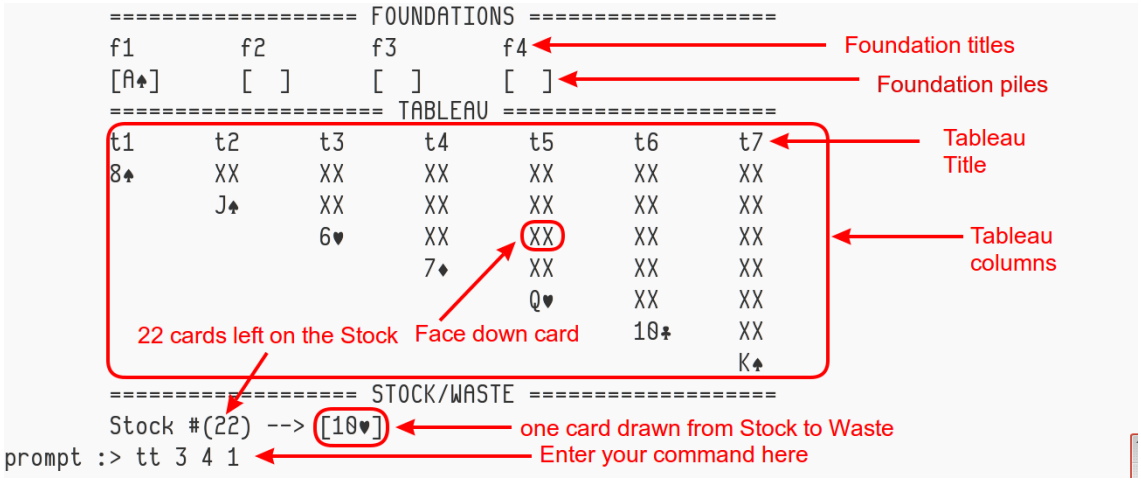
   That function has no parameters. It creates and initializes the tableau, foundation, stock and waste, and then returns them as a tuple, in that order. The Foundations will be a list of 4 empty lists, the Tableau will contain 28 cards in 7 vertical columns as described

before, the Stock is a deck and will contain the remaining 23 cards and there will be one card (top card from the Stock) on the Waste. *The order of dealing cards to the Tableau is precisely defined for solitaire* (which conveniently makes grading easier): To form the Tableau, seven piles need to be created. Starting from left to right, place the first card face up to make the first pile, deal one card face down for the next six piles. Starting again from left to right, place one card face up on the second pile and deal one card face down on piles three through seven. Starting again from left to right, place one card face up on the third pile and deal one card face down on piles four through seven. Continue this pattern until pile seven has one card facing up on top of a pile of six cards facing down. The remaining cards form the Stock pile. When starting out, the Foundation piles do not have any cards. After dealing the Tableau draw one card from the Stock pile for the Waste pile.

2. (Suggestion: at first simply print each data structure, e.g. `print(tableau)` in this function until you get the rest of the program working. Then come back and do the formatting last.)
   The program will use the following function to display the current state of the game:

   `display_game(foundation, tableau, stock, waste)` → `None`

   This function has four parameters – the data structure representing the Foundation, the Tableau, Stock and the Waste. The foundation is displayed first. A non-empty foundation pile will be displayed as the top card in the pile (i.e. last card moved to it); and an empty foundation will be displayed as **`[ ]`**. The tableau is displayed second; each column of the tableau will be displayed downwards as shown in the sample below. An empty column will be displayed by whitespace. For the case of Stock, this function will only display how many cards are left on the Stock like **`Stock #(k)`** following with an arrow sign **`-->`**, (here **`k`** stands for the number of cards left on the stock) then all the Waste cards will be displayed in a square bracket like **`[ card1, card2 … etc. ]`** where card1 is the bottom of the waste and card-n is the top of the waste, and commas separate the cards. The diagram below shows the display early in a game.



To enter command, there is a prompt that looks like **`prompt:>`** ; this is already implemented in the **`main()`** function.

3. This function below may be used to decide whether a movement from Tableau/Waste to Foundation is valid or not:

`valid_fnd_move(src_card, dest_card)` → **`True or False`**

The function has two parameters, **`src_card`** and **`dest_card`**. The card that you are trying to place is the **`src_card`** and the card that you are trying to place onto is the **`dest_card`**. The function will return True if the move is valid; False otherwise. This function will return **`True`** if the **`src_card`** and **`dest_card`** have the *same* suit and the rank of the **`src_card`** is 1 greater than that of the **`dst_card`**. (Two conditions may be checked here or elsewhere: [1] the **`src_card`** is invalid, e.g. does not exist, or [2] if the Foundation pile is empty, only an ace is valid for placement on the Foundation.)

4. This function below may be used to decide whether a card (or a set of cards') movement from Tableau/Waste to another Tableau is valid or not:

`valid_tab_move(src_card, dest_card)` → **`True or False`**

The function has two parameters, **`src_card`** and **`dest_card`**. The card that you are trying to place is the **`src_card`** and the card that you are trying to place onto is the **`dest_card`**. The function will return True if the move is valid; False otherwise. This function will return **`True`** if the rank of the **`dest_card`** is 1 greater than that of the **`src_card`**, and they are from *different* suits. (Two conditions may be checked here or elsewhere: [1] the **`src_card`** is invalid, e.g. does not exist, or [2] if the destination Tableau column is empty, any card (irrespective of rank or suit) is valid for placement.)

5. This function below will implement a movement of a card from Tableau to Foundation:

   `tableau_to_foundation(tab, fnd)` → **`None`**

   This function has two parameters, both are lists. The first one is the source Tableau column (i.e. from which you are moving a card) and the second parameter is the destination Foundation pile. This function moves a single card and this function calls **`valid_fnd_move()`** to decide if a movement is valid or not. If there is an invalid move, it will raise a **`RuntimeError`** exception. This function returns nothing. When a user enters a **`tf x y`** command, this function will be used. You can only move a face-up card. When you move a card from Tableau to Foundation, if the "next" card on that Tableau column exists and is face down, it needs to be flipped.

6. This function below will implement a movement of a card (a set of cards) from one Tableau column to another Tableau column:

   `tableau_to_tableau(tab1, tab2, n)` → **`None`**

   This function has three parameters, first two are lists and the third is an integer. The first one is the Tableau column from which you are moving a card (a set of cards) from and the second parameter is the destination Tableau pile. This function moves **`n`** (**`n >= 1`**) number of cards and this function calls **`valid_tab_move()`** to decide if such a movement is valid or not. If there is an invalid move, it will raise a **`RuntimeError`** exception. This function returns nothing. When a user enters a **`tt x y n`**, this function will be used. You can only move a face-up card (or a set of cards). When you move a card (or a set of cards) from source Tableau to destination Tableau column, if the "next" card on the source Tableau column exists and is face down, it needs to be flipped.

7. This function below will implement a movement of a card from Waste to Foundation:

   `waste_to_foundation(waste, fnd, stock)` → **`None`**

   This function has three parameters, all of them are lists. The first one is the Waste pile from which you are moving a card from; the second parameter is the destination Foundation pile and the third parameter is the Stock pile (the Stock parameter is not needed—a leftover from an earlier version). This function moves a single card and this function calls **`valid_fnd_move()`** to decide if a movement is valid or not. If there is an invalid move, it will raise a **`RuntimeError`** exception. This function returns nothing. This function only moves the top card from the Waste pile. When a user enters a **`wf x`** command, this function will be used.

8. This function below will implement a movement of a card from Waste to Tableau column:

   `waste_to_tableau(waste, tab, stock)` → `None`

   This function has three parameters, all of them are lists. The first one is the Waste pile from which you are moving a card from; the second parameter is the destination Foundation pile and the third parameter is the Stock pile (the Stock parameter is not needed—a leftover from an earlier version). This function moves a single card and this function calls **`valid_tab_move()`** to decide if a movement is valid or not. If there is an invalid move, it will raise a **`RuntimeError`** exception. This function returns nothing. This function only moves the top card from the Waste pile. When a user enters a **`wt x`** command, this function will be used.

9. This function will be used when user enters an **`sw`** command:

   `stock_to_waste(stock, waste)` → `None`

   This function will move the top card from the Stock to Waste. If such movement is not possible, it will raise a **`RuntimeError`** exception. This function returns nothing.

10. This function will be used to decide if a game was won:

    `is_winner(foundation)` → `True/False`

    A game is a win if all the Foundation piles have 13 cards and the top card on each of the pile is a King. Or you can change this function to **`is_winner(tableau, stock, waste)`** and it will return **`True`** if all the Tableau columns are empty (and Stock and Waste are also empty). This function returns either **`True`** or **`False`**.

Some Mandatory Stuff:

1. The **`display_game()`** function must generate the output similar to the example.
2. You must use the **`try-except`** block to handle errors (the only try-except block needed is included in the sample), each error message must explain exactly what type of errors a user is making. There are many errors to consider—we do not enumerate them for you. Please see the Bad Session example for this purpose.
3. Use the **`RuntimeError`**'s custom message option to throw and handle errors (I did this for *all* errors). You put the error message in quotes and when the exception is caught in the main your message in quotes is passed as `error_message` in the `except` part and is then printed.

```
if everything looks fine:
```

```
        do whatever necessary
else:
        raise RuntimeError("Error: invalid command because of …")
```

Assignment Notes and Tips:

1. Before you begin to write any code, play with the game at http://politaire.com/thumbandpouch and look over the sample interaction below to be sure you understand the rules of the game and how you will simulate the game.

2. We provide a module called **`cards.py`** that contains a Card class and a Deck class. Your program must use this module (**`import cards`**). You should also understand this file line by line and do not modify this file!

3. Laboratory Exercise #12 demonstrates how to use the cards module. Understanding those programs should give you a good idea how you can use the module in your game.

4. We have provided a framework named **`proj10.py`** to get you started. Using this framework is mandatory. It runs as is, but does nothing useful. Gradually replace the "stub" code (marked with **pass** and comments) with your own code. (Delete the stub code.) Modify and test functions one at a time—do not move on to a new function until existing functions are correct.

5. Displaying the Tableau in columns is tricky. Start by implementing a very simple display function: You can easily label and display the foundation on one line, and the tableau using seven lines, with each line displaying the cards in a column. Once you have the game logic working properly, modify your display function to display the current game state as described in the specification.

6. TT command: Moving a pile of length n > 1 is harder than moving n = 1. First write your code to move exactly one card (n = 1). Once moving one card is working and the rest of the program is working, incorporate moving piles of length n > 1.

7. The list `pop()` command is useful for this assignment (but its use is not required). I found `extend()` useful (in addition to append()).

8. One thing that can help testing is to turn off shuffling when the game starts.

9. For debugging it can be useful to start and quit the program so data structures representing Tableau, Foundation, Stock, and Waste are created. Then in the Python shell you can create individual cards and add them to your data structures. You can then (also in the Python shell) call functions to test them individually:

```
In [44]: c1 = cards.Card(2,3)

In [45]: print(c1)
2♥

In [46]: tab
Out[46]:
[[K♠],
 [XX, 6♠],
 [XX, XX, K♥],
 [XX, XX, XX, 8♥],
 [XX, XX, XX, XX, 4♥],
 [XX, XX, XX, XX, XX, A♥],
 [XX, XX, XX, XX, XX, XX, Q♦]]

In [47]: tab[3].append(c1)

In [48]: tab
Out[48]:
[[K♠],
 [XX, 6♠],
 [XX, XX, K♥],
 [XX, XX, XX, 8♥, 2♥],
 [XX, XX, XX, XX, 4♥],
 [XX, XX, XX, XX, XX, A♥],
 [XX, XX, XX, XX, XX, XX, Q♦]]
```

10. The coding standard for CSE 231 is posted on the course website:

    http://www.cse.msu.edu/~cse231/General/coding.standard.html

    Items 1-9 of the Coding Standard will be enforced for this project.

11. Your program should not use any global variables inside of functions. That is, all variables used in a function body must belong to the function's local name space. The only global references will be to functions and constants.

12. Your program must contain the functions listed in the previous sections; you may develop additional functions, as appropriate.

Sample Output:

Because the interactions are lengthy, they are saved in separate files; they are available on the project folder. They are partial games—neither goes to completion.

1. **`good-session.txt`**: an example game session with few errors. A modified cards.py file is used with shuffle turned off so the game is played the same way each time. Also, a specific set of cards is used when creating a deck. That special cards file is named cards1.py.
2. **`bad-session.txt`**: an example game session demonstrating more error messages.

These are plain text files, please use notepad/wordpad (textedit in osx) to open them.

==========

Educational Research

When you have completed the project insert the 5-line comment specified below.

For each of the following statements, please respond with how much they apply to your experience completing the programming project, on the following scale:

1 = Strongly disagree / Not true of me at all

2

3

4 = Neither agree nor disagree / Somewhat true of me

5

6

7 = Strongly agree / Extremely true of me

****Please note that your responses to these questions will not affect your project grade, so please answer as honestly as possible.****

Q1: Upon completing the project, I felt proud/accomplished

Q2: While working on the project, I often felt frustrated/annoyed

Q3: While working on the project, I felt inadequate/stupid

Q4: Considering the difficulty of this course, the teacher, and my skills, I think I will do well in this course.

Please insert your answers into the bottom of your project program as a comment, formatted exactly as follows (so we can write a program to extract them).

Questions

Q1: 5

Q2: 3

Q3: 4

Q4: 6