CSE 231 **Fall 2015**

Computer Project #9

Assignment Overview

This assignment focuses on the design, implementation and testing of a Python program which uses an instructor-supplied module to play a card game, as described below.

It is worth 60 points (6% of course grade) and must be completed no later than 11:59 PM on Monday, November 23.

Assignment Deliverables

The deliverable for this assignment is the following file:

`proj09.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.

Assignment Background

Aces Up is a popular solitaire card game which is played by one person with a standard 52-card deck of cards. The rules and a tutorial video are available at:

http://worldofsolitaire.com/

Under the "Solitaire" tab, click on "Select Game…" and "Aces Up".

Your program will allow the user to play a simplified version of Aces Up, with the program managing the game. The game rules are given below.

Game Rules

1. Start: The game is played with one standard deck of 52 cards. The deck is shuffled and becomes the initial *stock* (pile of cards). The cards in the stock are placed face down.

Four cards are then dealt from the stock, left to right, one to each column of a four-column *tableau*. During play, additional cards will be added to the tableau, but it starts with just one card in each column. All tableau cards are visible.

The game also has a *foundation*, which is a single pile of cards. The foundation starts out empty and is filled during play.

2. Goal: The game is won when the stock is empty and the only cards left in the tableau are the four aces.

3. Moves: A player can move only one card at a time and the moved card must be the bottom card of a tableau column.

The card at the bottom of a column can be moved to the foundation if a higher rank card of the same suit is at the bottom of another column. Note that aces are high in this game—that is, the rank of an ace is greater than the rank of any other card.

If a column of the tableau becomes empty, any card at the bottom of another column can be moved to the empty column.

No other moves are permitted.

4. Deal: A player can choose to deal instead of moving a card. In this case, the top four cards are dealt from the stock, left to right, with one card placed at the bottom of each column of the tableau. If the stock has fewer than four cards, the cards are dealt to tableau columns until the stock is empty.

Assignment Specifications

You will develop a program that allows the user to play Aces Up according to the rules given 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 on the project directory (**`sample_interaction.pdf`**).

1. The program will recognize the following commands (upper or lower case):

`D`	Deal four cards from the stock to the tableau
`F x`	Move card from Tableau column **`x`** to the Foundation.
`T x y`	Move card from Tableau column **`x`** to empty Tableau column **`y`**.
`R`	Restart the game (after shuffling)
`H`	Display the menu of choices
`Q`	Quit

where **`x`** and **`y`** denote column numbers, and the columns 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`**" (or "**`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.

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

`init_game() → (stock, tableau, foundation)`

That function has no parameters. It creates and initializes the stock, tableau, and foundation, and then returns them as a tuple, in that order.

3. The program will use the following function to deal cards to the tableau:

`deal_to_tableau( stock, tableau ): → None`

That function has two parameters: the data structure representing the stock and the data structure representing the tableau. It will deal a card from the stock to each column of the tableau, unless the stock has fewer than 4 cards; in which case it will just deal a card to consecutive columns until the stock is empty.

4. The program will use the following function to display the current state of the game:

`display( stock, tableau, foundation ) → None`

That function has three parameters: the data structure representing the stock, the data structure representing the tableau, and the data structure representing the foundation. A header line will be displayed labeling the stock, tableau, and foundation.

Under the "stock" header, a non-empty stock will be displayed as "**XX**", and an empty one will be displayed as whitespace.

Under the "tableau" header, each column of the tableau will be displayed in order: a non-empty column by the cards in the column, in order, from the first card moved that is still left in the column (top) to the last card moved that is still left in the column (bottom); and an empty column will be displayed by whitespace.

Under the "foundation" header, a non-empty foundation will be displayed as the top card in the foundation (i.e. last card moved to it); and an empty foundation will be displayed as whitespace.

5. The program will use the following function to prompt the user to enter an option and return a representation of the option designed to facilitate subsequent processing.

`get_option()→ list`

That function takes no parameters. It prompts the user for an option and checks that the input supplied by the user is of the form requested in the menu. If the input is not of the required form, the function prints an error message.

The function returns a list as follows:

- **[]**, if the input is not of the required form
- **['D']**, for deal
- **['F', x]**, where **x** is an **int**, for moving a card to the foundation
- **['T', x, y]**, where **x** and **y** are **int**'s, for moving a card within the tableau
- **['R']**, for restart
- **['H']**, for displaying the menu
- **['Q']**, for quit

6. The program will use the following function to determine if a requested move to the foundation is valid:

validate_move_to_foundation(tableau, from_col) → bool

That function has two parameters: the data structure representing the tableau and an **int** indicating the column whose bottom card should be moved. The function will return **True,** if the move is valid; and **False**, otherwise. In the latter case, it will also print an appropriate error message.

7. The program will use the following function to move a card from the tableau to the foundation:

move_to_foundation(tableau, foundation, from_col) → None

That function has three parameters: the data structure representing the tableau, the data structure representing the foundation, and an **int** indicating the column whose bottom card should be moved. If the move is valid, the function will update the tableau and foundation; otherwise, it will do nothing to them.

8. The program will use the following function to detrmine if a requested move to within the tableau is valid:

validate_move_within_tableau(tableau, from_col, to_col) → bool

That function has three parameters: the data structure representing the tableau, an **int** indicating the column whose bottom card should be moved, and an **int** indicating the column the card should be moved to. The function will return **True,** if the move is valid; and **False**, otherwise. In the latter case, it will also print an appropriate error message.

9. The program will use the following function to move a card from the tableau to the foundation:

`move_within_tableau( tableau, from_col, to_col ) → None`

That function has three parameters: the data structure representing the tableau, an **`int`** indicating the column whose bottom card should be moved, and an **`int`** indicating the column the card should be moved to. If the move is valid, the function will update the tableau; otherwise, it will do nothing to it.

10. The program will use the following function to check if the game has been won:

`check_for_win( stock, tableau ) → bool`

That function has two parameters: the data structure representing the stock and the data structure representing the tableau. If returns **`True`**, if the stock is empty and the tableau contains only the four aces; and **`False`**, otherwise.

Assignment Notes

1. Before you begin to write any code, play with the provided demo program and look over the sample interaction supplied on the project website to be sure you understand the rules of the game and how you will simulate the demo program. The demo program is at http://worldofsolitaire.com/: Under the "Solitaire" tab, click on "Select Game…" and "Aces Up".

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`**). *Do not modify this file!* This is a generic module for any card game and happens to be implemented with "aces low" (having a rank of 1). Your game requires aces to have a rank higher than any other card. Your program must implement "aces high" *without modifying the* **`cards`** *module.*

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 **`proj09_skeleton.py`** to get you started. *Using this framework is mandatory*. Begin by copying or renaming it to **`proj09.py`**. Check that it compiles. Gradually replace the "stub" code (marked with comments) with your own code. (Delete the stub code.)

5. Displaying the tableau in columns is tricky. Start by implementing a very simple **`display`** function: You can easily label and display the stock and foundation on one line, and the tableau using four 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 above in the specification for function **`display`**. (Displaying the tableau in columns instead of rows will cost only a small number of points compared to enforcing the rules of the game.)

6. 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.

7. Your program may 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.

8. Your program may not use any nested functions. That is, you may not nest the definition of a function inside another function definition.

9. Your program must contain the functions listed above; you may develop additional functions, as appropriate.