Klondike
CMPUT 175 Assignment 2

Due Date: November 19

Klondike is a solitaire card game. In the U.S. and Canada, Klondike is the best-known solitaire card game, to the point that the term "Solitaire", in the absence of additional qualifiers, typically refers to Klondike and is considered its other name. [Wikipedia]

Your assignment is to produce a program that plays Klondike.

Rules

[This text is partially based on Wikipedia.]

Klondike is played with a standard 52-card deck, without Jokers. After shuffling, seven fanned **Piles** of cards are laid. From top to bottom, each **Pile** contains one more card than the last. In each **Pile**, the cards are face-down except for the topmost card, which is turned face-up. Note that in most implementations of Klondike, the piles are displayed from left to right.

The **Piles** (numbered **1** to **7**) look as follows, where “??” indicates an (unknown) face-down card.

```
  PILE-1 [ 9s ]
  PILE-2 [ Qd ?? ]
  PILE-3 [ 5s ?? ?? ]
  PILE-4 [ Td ?? ?? ?? ]
  PILE-5 [ 3d ?? ?? ?? ?? ]
  PILE-6 [ 8d ?? ?? ?? ?? ?? ]
  PILE-7 [ 4d ?? ?? ?? ?? ?? ?? ]
```

The remaining 24 cards form the **Stock** and are placed face-down. The top card is turned face-up. Cards are eventually discarded from the top of **Stock** and put face-down in the **Discard** pile.

There is a pile for each of the four suits (**Spades**, **Hearts**, **Diamonds**, **Clubs**). These are built up by suit from Ace (low in this game) to King (“A”,”2”,”3”,...,”9”,”T”,”J”,”Q”,”K”).

In the following, the 7 of clubs is the top card in **Stock**. There are no cards in **Discard**, **Spades**, **Hearts**, **Clubs** and **Diamonds**.

```
   Stock [ 7c ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ]
 Discard [ ]
  Spades [ ]
  Hearts [ ]
Diamonds [ ]
   Clubs [ ]
  PILE-1 [ 9s ]
  PILE-2 [ Qd ?? ]
  PILE-3 [ 5s ?? ?? ]
  PILE-4 [ Td ?? ?? ?? ]
  PILE-5 [ 3d ?? ?? ?? ?? ]
  PILE-6 [ 8d ?? ?? ?? ?? ?? ]
  PILE-7 [ 4d ?? ?? ?? ?? ?? ?? ]
```

The goal of the game is to play a sequence of valid moves (see below) so that all the cards end up in the suits. Thus, each of the four suits is a sequence from Ace to King. This is the winning state:

```
   Stock [ ]
 Discard [ ]
```

```
   Spades [ Ks Qs Js Ts 9s 8s 7s 6s 5s 4s 3s 2s As ]
   Hearts [ Kh Qh Jh Th 9h 8h 7h 6h 5h 4h 3h 2h Ah ]
 Diamonds [ Kd Qd Jd Td 9d 8d 7d 6d 5d 4d 3d 2d Ad ]
    Clubs [ Kc Qc Jc Tc 9c 8c 7c 6c 5c 4c 3c 2c Ac ]
   PILE-1 [ ]
   PILE-2 [ ]
   PILE-3 [ ]
   PILE-4 [ ]
   PILE-5 [ ]
   PILE-6 [ ]
   PILE-7 [ ]
```

The user moves cards around according to the following rules. In the text below, the following example is used:

```
    Stock [ Ac ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ]
  Discard [ ]
   Spades [ ]
   Hearts [ ]
 Diamonds [ ]
    Clubs [ ]
   PILE-1 [ 7c 8d 9s ]
   PILE-2 [ Qd ?? ]
   PILE-3 [ 5s ?? ?? ]
   PILE-4 [ Td ?? ?? ?? ]
   PILE-5 [ 4h ?? ?? ?? ]
   PILE-6 [ As ?? ?? ?? ]
   PILE-7 [ 2c 3d 4d ?? ?? ?? ?? ?? ?? ]
```

Cards can move from **Stock** or any of the **Piles** to the four suits, as long as the suits are built up in order (Ace to King). In the example, The Ace of clubs can move from **Stock** to **Clubs**. Also, the Ace of spades in **Pile 6** can move to **Spades**. Further, the 2 of clubs in **Pile 7** can move to **Clubs** (after the Ace has moved there). If after a move the top card in the **Stock** or a **Pile** is face-down, then it is turned face-up.

```
    Stock [ Js ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ]
  Discard [ ]
   Spades [ As ]
   Hearts [ ]
 Diamonds [ ]
    Clubs [ 2c Ac ]
   PILE-1 [ 7c 8d 9s ]
   PILE-2 [ Qd ?? ]
   PILE-3 [ 5s ?? ?? ]
   PILE-4 [ Td ?? ?? ?? ]
   PILE-5 [ 4h ?? ?? ?? ]
   PILE-6 [ 6c ?? ?? ]
   PILE-7 [ 3d 4d ?? ?? ?? ?? ?? ?? ]
```

The **Piles** can have cards placed on them, as long as the cards form a decreasing sequence. In the example, the 4 of hearts in **Pile 5** can move to **Pile 3** – a 4 can go on top of a 5. It's not just single cards that can move – a sequence of descending cards can move. In the example, instead of moving the 4 of hearts from **Pile 5** to **Pile 3**, the sequence [4 diamonds, 3 diamonds] in **Pile 7** can move to **Pile 3**, creating a sequence of 4 cards [5, 4, 3].

Note: the official rules of Klondike require sequences to alternate card colours. That means a red card (heart, diamond) can go on a black card (spade, club) and vice versa. For this assignment we will ignore the colour rule. This allows a higher percentage of games to be winnable.

A valid move is from **Pile 1** [9 spades, 8 diamonds, 7 clubs] to **Pile 4**.

```
   Stock [ Js ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ]
 Discard [ ]
  Spades [ As ]
  Hearts [ ]
Diamonds [ ]
   Clubs [ 2c Ac ]
  PILE-1 [ ]
  PILE-2 [ Qd ?? ]
  PILE-3 [ 3d 4d 5s ?? ?? ]
  PILE-4 [ 7c 8d 9s Td ?? ?? ?? ]
  PILE-5 [ 4h ?? ?? ?? ]
  PILE-6 [ 6c ?? ?? ]
  PILE-7 [ 7s ?? ?? ?? ?? ?? ]
```

The top **Stock** card can also move to a **Pile**. If this happens, then the next card in **Stock** is turned face-up.

Play can continue. For example, you might want to move:

- Move from **Stock** [J spades] to **Pile 2**.
- Move from **Pile 6** [6 diamonds] to **Pile 7** (or to **Pile 4**).
- Move from **Pile 2** [Q diamonds, J spades] to **Pile 6**.

```
   Stock [ 5d ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ]
 Discard [ ]
  Spades [ As ]
  Hearts [ ]
Diamonds [ ]
   Clubs [ 2c Ac ]
  PILE-1 [ ]
  PILE-2 [ Qc ]
  PILE-3 [ 3d 4d 5s ?? ?? ]
  PILE-4 [ 7c 8d 9s Td ?? ?? ?? ]
  PILE-5 [ 4h ?? ?? ?? ]
  PILE-6 [ Js Qd Kh ?? ]
  PILE-7 [ 6c 7s ?? ?? ?? ?? ?? ]
```

One more rule: when a **Pile** is empty, only a King (or sequence of cards headed by a King) can move to the empty **Pile**.

- Move **Pile 6** [King hearts, Q diamonds, J spades] to **Pile 1**.
- Move **Stock** [5 diamonds] to **Pile 7**.
- Move from **Pile 5** [4 hearts] to **Pile 7**.
- Move from **Pile 4** (or **Pile 6**) [T diamonds, 9 spades, 8 diamonds, 7 clubs] to **Pile 1**.
- Move from **Pile 4** [7 diamonds] to **Pile 5**.

```
   Stock [ 8h ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ]
 Discard [ ]
  Spades [ As ]
  Hearts [ ]
Diamonds [ ]
   Clubs [ 2c Ac ]
  PILE-1 [ 7c 8d 9s Td Js Qd Kh ]
  PILE-2 [ Qc ]
  PILE-3 [ 3d 4d 5s ?? ?? ]
  PILE-4 [ Jd ?? ]
  PILE-5 [ 7d 8s ?? ?? ]
  PILE-6 [ Th ]
  PILE-7 [ 4h 5d 6c 7s ?? ?? ?? ?? ?? ]
```

Although there are some moves available (e.g., from **Pile 7** [6 clubs, 5 diamonds, 4 hearts] to **Pile 5**), these moves do not advance the game – they just shuffle the position of cards.

When the player has exhausted their useful moves, then the top three cards (or less if there are only 1 or 2 cards left) are moved from **Stock** to **Discard** (face-down in the same order). The top **Stock** card is turned face-up, and play continues.

When **Stock** is empty, the **Discard** cards move to **Stock**, the top **Stock** card is turned face-up, and play continues.

If all four of the suit piles (**Spades**, **Hearts**, **Diamonds**, **Clubs**) and completely built up (from Ace to King in order), then the game is won. If the player can no longer make any meaningful moves, the game is considered lost.

Here we move 3 cards from **Stock** to **Discard**.

```
   Stock [ Tc ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ]
 Discard [ ?? ?? ?? ]
  Spades [ As ]
  Hearts [ ]
Diamonds [ ]
   Clubs [ 2c Ac ]
  PILE-1 [ 7c 8d 9s Td Js Qd Kh ]
  PILE-2 [ Qc ]
  PILE-3 [ 3d 4d 5s ?? ?? ]
  PILE-4 [ Jd ?? ]
  PILE-5 [ 7d 8s ?? ?? ]
  PILE-6 [ Th ]
  PILE-7 [ 4h 5d 6c 7s ?? ?? ?? ?? ?? ]
```

Get a deck of cards, deal out a hand, and play the game. If you do not have a deck of cards, try playing one of the online versions of the game (but remember that the online versions use the alternating colour rule). This is a good way to familiarize yourself with the rules.

Commands

Your program will accept the following commands. Each command has 0, 1, or 2 parameters.

Commands are read from the keyboard using `input()`. All output is done using `print()` – except for the **save** command (see below).

# File commands
load filename
save filename

Save/load a game state. Here is the format for load/save:

```
Stock [ Tc+ Qh- 5c- Ad- 2d- 9d- 3h- Jh- 5h- 9h- 9c- 6d- 2h- Ks- 3c- Ah- 8c- ]
Discard [ Qs- Kd- 8h- ]
Spades [ As+ ]
Hearts [ ]
Diamonds [ ]
Clubs [ 2c+ Ac+ ]
PILE-1 [ 7c+ 8d+ 9s+ Td+ Js+ Qd+ Kh+ ]
PILE-2 [ Qc+ ]
PILE-3 [ 3d+ 4d+ 5s+ 6h- Ts- ]
PILE-4 [ Jd+ Jc- ]
PILE-5 [ 7d+ 8s+ 4c- 2s- ]
```

```
PILE-6 [ Th+ ]
PILE-7 [ 4h+ 5d+ 6c+ 7s+ Kc- 7h- 3s- 4s- 6s- ]
```

There are 13 lines, one for each pile of cards. The cards are listed from the top of the deck to the bottom. "+" indicates the card is face-up; "-" indicates it is face-down. A good implementation of **save** would use the `__repr()__` function to do most of the work.

# Game control commands

done

Exit from your **Klondike** program (without saving the game state).

discard

Move three (or less, if **Stock** has only 1 or 2 cards left) from **Stock** to **Discard**. The cards in **Discard** are face-down. The top **Stock** card is turned face-up.

reset

The **Stock** must be empty. Move the **Discard** cards into **Stock**, leaving **Discard** empty. The **Stock** cards are in the order in which they were discarded. The top **Stock** card is turned face-up.

# Document commands

board

Display the current game state (see the example in this file and the sample files provided).

cheat

This is the same as **print**, except all hidden cards are displayed. See the sample files provided.

comment text

All the **text** is displayed as a comment. See the sample files provided.

# Game play

move from to

The parameters are "**stock**", "**suit**", and a **Pile** number ("**1**"-"**7**"). There are four cases to consider:

from = **stock** and **to** = **suit**:

Move the top **Stock** card to the appropriate suit, if legal. The card on top of **Stock** is turned face-up.

from = **stock** and **to** = **1**-**7**:

Move the top **Stock** card to the specified **Pile**, if legal. A King can move to an empty **Pile**. The card on top of **Stock** is turned face-up.

from = **1**-**7** and **to** = **suit**:

Move the top card in the specified **Pile** to the appropriate suit, if legal. If the top card in the **Pile** is face-down, it is turned face-up.

from = **1**-**7** and **to** = **1**-**7**

Move a sequence of one or more cards (in consecutive order) from the specified from-**Pile** to the specified to-**Pile**, if legal. If the top card in the from-**Pile** is face-down, it is turned face-up. A sequence headed by a King can move to an empty to-**Pile**.

All the above scenarios can be seen in the example files provided.

Design

*1. Create a **Card** class, where each card in a deck is an object.*

Ensure your class contains the following methods:

- `__init__(self,suit,rank)`
- `__str__(self)`
- `__repr__(self)`
- `isvisibleCard(self)` – returns whether the card is visible (face-up or face-down) (i.e., `True` or `False`).
- `faceupCard(self,visible)` – sets the visibility status (`True` or `False`) of a card.
- `rankCard()` – returns the rank of a card.
- `suitCard()` – returns the suit of a card.

You cannot add additional methods to the `Card` public interface. However, you can create additional "helper" methods that are only called within the `Card` class.

*2. Create a **Deck** class, which holds **Card** objects.*

The implementation should be based on the **Stack** class used in class (feel free to use the code from class; no attribution needed). Your solution will need 13 **Deck** objects (**Stock**, **Discard**, **Spades**, **Hearts**, **Diamonds**, **Clubs**, **Pile-1**...**Pile-7**).

- The number of cards in **Stock** is always decreasing (by one at a time, or up to 3 when a **discard** occurs), until it is **reset**. All moves affecting the **Stock** (except **reset**) remove a card.
- The number of cards in **Discard** is always growing (usually by three at a time), until it is **reset**. A **discard** moves cards from **Stock** to **Discard**.
- The four suits start empty and grow by one card at a time.
- The **Piles** can grow and shrink. Be careful! A move of, say, four cards from **Pile-7** to **Pile-3** is not a natural **Stack** operation. However, there are simple ways to perform this operation without violating the **Stack** abstraction.

Your **Deck** class should include two private variables: **name**, a string giving the name of the **Deck**, and **cards**, a list of card objects in the **Deck**.

Ensure your class contains the following methods:

- `__init__(self,name)`
- `__str__(self)` – this can be used as the basis for the **print** and **cheat** commands
- `__repr__(self)` – this can be used as the **save** file format
- `nameDeck()` – return the name of the **Deck** as a string.
- `sizeDeck()` -- return the number of cards in the **Deck**.
- `isemptyDeck()` – return whether the **Deck** is empty (`True` or `False`).
- `pushDeck(card)` – put card on top of the **Deck**.
- `popDeck()` – return the card on top of the **Deck** and remove it.

- `peekDeck()` – return the card on top of the **Deck** without removing it.

You cannot add additional methods to the **`Deck`** public interface. However, you can create additional “helper” methods that are only called within the **`Deck`** class.

As you know, there are multiple ways to implement a **Stack**. Please use an implementation that has O(1) time for `pushDeck()` and `popDeck()`.

3. Use Python’s `assert` and `exception` capabilities.

Use these for error handling of user input and file I/O. Some of the situations you need to handle will generate messages as follows:

```
"%s: could not open file" % (command)
"%s: format error in file" % (command)
"%s: illegal command" % (command)
"%s: incorrect number of arguments" % (command)
"%s: arguments incorrect" % (command)
"%s: illegal move" % (command)
```

There are other exceptional situations you will need to address. They should follow the same format as above:

```
# You might want to include other values to print
"%s: <<your message>>" % (command)
```

Restrictions for this assignment (same as Assignment 1): You cannot use `break`/`continue`, and you cannot `import` any modules (except in one case – see Extensions below). Doing so will result in deductions.

Testing

You will be provided with the following files:

- **Game1-input.txt** – a series of commands for your program to execute
- **Game1-start.txt** – a starting position for **load** (the results of a **save** command)
- **Game1-log.txt** – the results of running your program with the input from **Game1-input.txt**

When your program is run with the commands from **Game1-input.txt,** the output should be identical to that given in **Game1-log.txt**.

Your program will need extensive testing. The **cheat** command has been added to the user interface of your main program to help support your debugging and testing efforts.

It is your responsibility to thoroughly test your program, including potential error scenarios.

Rubric

- Code quality and adherence to the specifications: 15%

- Card class: 10%
- Deck class: 15%
- Use of asserts/exceptions: 10%
- File I/O: 10%
- Printing (board and cheat): 10%
- Main program (e.g., move correctness): 20%
- Program control and menu input validation: 10%
- Bonus extension (see below): 10%

Hints for Getting Started

- Implement the Card class and test it.
- Implement the `__init__`, `__str__`, and `__repr__` methods for **Deck**. As you implement the other methods for **Deck**, be sure to test them as you go.
- Implement the **load** command.
- Implement the **print** and **cheat** commands to verify that you are **load**ing correctly.
- Only then, start tackling the **move** command.

Program Extensions

There are obvious extensions to this assignment that you can consider implementing:

1. For bonus marks... add the alternating colour rule to the **Pile** sequences.
2. For your own enjoyment, add a **shuffle** command which starts a new game and shuffles the cards. You can `import random` and using its random number generator to help shuffle the cards.
3. For your own enjoyment, add an AI (artificial intelligence) component to the program so that your program decides what moves to play. Can your AI play Klondike better than you?

Submission Instructions

Please follow these instructions to correctly submit your solution:

- All your code should be contained in a single Python file: **klondike.py**.
- Make sure that you include your name (as author) in a header comment at the top of **klondike.py**, along with an acknowledgement of any collaborators/references.
- Please submit your **klondike.py** file via eClass before the due date/time.
- Do not include any other files in your submission.

- **Late submissions will not be accepted**. You can make as many submissions as you like before the deadline – only your last submission will be marked. Submit early; submit often.

REMINDER: Plagiarism will be checked for

Just a reminder that, as with all submitted assessments in this course, we use automated tools to search for plagiarism. In case there is any doubt, you **CANNOT** post this assignment (in whole or in part) on a website like Chegg, Coursehero, StackOverflow or something similar and ask for someone else to solve this problem (in whole or in part) for you. Similarly, you cannot search for and copy answers that you find already posted on the Internet. You cannot copy someone else’s solution, regardless of whether you found that solution online, or if it was provided to you by a person you know. **YOU MUST SUBMIT YOUR OWN WORK**.