Computer Science

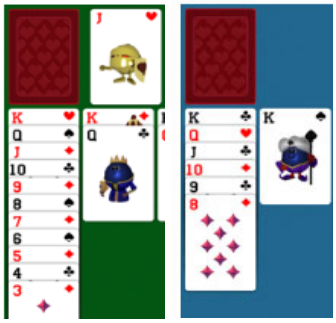
COMPSCI 130

Assignment

Due Date

Due: **11:59 pm 27th May 2022**
Worth: **This assignment is marked out of 18 and is worth 6% of your final mark.**

Klondike Solitaire is a game which is played with a standard 52-card deck. An example is shown as below:



In this assignment, you will implement a simplified version of the Klondike Solitaire game. The simplified version of the game is played with a list of cards. The ultimate objective is to place every card from the pack onto one of the foundation piles. There are only two types of piles in the simplified Klondike Solitaire game: "stock" and "foundation". The game consists of one stock (hand) pile and two foundation piles.

The simplified game rules

When the game starts, the stock pile begins with a shuffled deck of cards. Each card in the deck is represented by a unique number, and these numbers range from 0 to `n-1`, where `n` is the total number of cards. The game starts with one upturned card and *n*-1 downturned cards in the "stock" pile. The two "foundation" piles do not contain any cards. An example is shown below (where "S" represents the "stock" pile and "1" and "2" represent the two "foundation" piles):

```
S: 1 * * * * *
1:
2:
```

There are **two** valid types of moves using the stock pile. Note: in the examples below, all cards are shown upturned for illustration (but in practice only the first card in the stock pile would be shown upturned to the player).

Move 1: You can always move a card from the **rear** of the stock pile to the **front** of the stock pile. This move is always valid.

rear

```
S: 1 3 5 0 2 4
1:
2:
```

front

```
S: 3 5 0 2 4 1
1:
2:
```

Move 2: You can move a card from the **rear** of the stock pile to the **front** of either of the foundation piles. This move is valid only when the number of the card (*x*) at the top of the source pile (i.e. stock pile) is one less than the number of the card (*y*) at the front of the destination pile (i.e. foundation pile). (i.e. y = x + 1)

The following illustrates a *valid* move which moves a card from the **rear** of the stock pile to the **front** of the **first** foundation pile (the left hand column shows the card before the move, and the right hand column shows the cards after the move).

```
S: 2 4 1 0
1: 3
2: 5
```

```
S: 4 1 0
1: 3 2
2: 5
```

And the following illustrates a *valid* move which moves a card from the **rear** of the stock pile to the **front** of the **second** foundation pile. This is valid as the foundation pile is empty.

S: **4** 1 0 5 1: 3 2 2:	S: 1 0 5 1: 3 2 2: **4**

The following illustrates an ***invalid*** move as 5 is not equal to 2 + 1.

S: **2** 4 1 0 1: 3 2: 5	S: 2 4 1 0 1: 3 2: 5 **2 #invalid**	S: 2 4 1 0 1: 3 2: 5

For foundation piles, there is **one** type of valid move. You can move the whole pile of cards from one foundation pile to the front of the other foundation pile. This move is valid only when the number of the card (x) on the rear of the source foundation pile is one less than the number of the card (y) at the front of the destination foundation pile (i.e. $y = x + 1$).

The following illustrates a *valid* move which moves all cards from the first foundation pile to the second foundation pile

S: 1 0 1: **3 2** 2: 5 4	S: 1 0 1: 2: 5 4 **3 2**

And the following illustrates a valid move which moves all cards from the first foundation pile to the second foundation pile. This is *valid* as the second foundation pile is empty.

S: 5 4 1 0 1: **3 2** 2:	S: 5 4 1 0 1: 2: **3 2**

However, this is an *invalid* move as 2 is not equal to 5 + 1.

S: 1 0 1: 3 2 2: **5 4**	S: 1 0 1: 3 2 **5 4 #invalid** 2: 5 4	S: 1 0 1: 3 2 2: 5 4

Also, you cannot move cards from a foundation pile back to the stock pile.

S: 1: 3 2 2: **1 0**	S: **1 0 #invalid** 1: 3 2 2: 1 0	S: 1: 3 2 2: 1 0

Implementation Details:

You will need to implement 2 classes for this assignment:

1. The **`Stock`** class.
2. The **`Foundation`** class.

Each class is to be in a separate file (Stock.py and Foundation.py). You will also need to write the program file that plays the game. This file is to be named "YourUsernameGame.py" e.g. "abcd001Game.py" (where “abcd001” would be your username).

The Stock class

Our game will use an instance of the **`Stock`** class which represents the stock/hand pile. The **`Stock`** class contains the following methods. You should follow the instructions in CodeRunner to create the **`Stock`** class.

- **`def __init__(self, number_of_cards = 0)`**
- **`def size(self)`**
- **`def is_empty(self)`**
- **`def push_list(self, values)`**
- **`def display(self)`**
- **`def __str__(self)`**
- **`def add_front(self, item)`**
- **`def add_rear(self, item)`**
- **`def remove_front(self)`**
- **`def remove_rear(self`**
- **`def peek_front(self`**

- **`def peek_rear(self)`**
- **`def move(self, pile = None)`**

The Foundation class

Our game will use two instances of the **`Foundation`** class which represents a foundation pile. The **`Foundation`** class contains the following methods. You should follow the instructions in CodeRunner to create the **`Foundation`** class.

- **`def __init__(self)`**
- **`def size(self)`**
- **`def is_empty(self)`**
- **`def push_list(self, values)`**
- **`def __str__(self)`**
- **`def add_front(self, item)`**
- **`def add_rear(self, item)`**
- **`def remove_front(self)`**
- **`def remove_rear(self`**
- **`def peek_front(self`**
- **`def peek_rear(self)`**
- **`def move(self, pile)`**

The YourUsernameGame.py file

First, you should import the following modules into your program:

```
import random
random.seed(30)
from Stock import Stock
from Foundation import Foundation
```

We will use the main() function of the **`YourUsernameGame.py`** file to run our game. This function will call 4 helper functions:

1. The **`print_banner()`** function
2. The **`print_all_piles(stock, piles)`** function
3. The **`get_pile_number(prompt)`** function
4. The **`game_over(stock, piles, number_of_cards)`** function

The **`main()`** function will work as follows:

- Set the number of cards to 6.
- Create an instance of the stock pile using the number of cards defined above.
- Create two instances of the foundation pile.
- Print the game banner.
- Display the stock pile.
- While the game is not over:
    - Prompt the user to enter a source pile.
    - Prompt the user to enter a destination pile.
    - Move the card(s) as indicated by the user's input by calling the appropriate **`move()`** method in the **`Stock`** or **`Foundation`** class, printing the appropriate error message if a move is invalid.
    - If the move was successful, display all three piles.
- Print a congratulations message at the end.

When the game begins, the program prints a game banner and the numbers in the stock pile:

```
CS130 Assignment - Simplified Klondike Solitaire
================================================
S: 1 * * * * *
```

The user is prompted to enter a source pile number and then prompted to enter a destination pile number. 0 represents the stock pile, 1 represents the first foundation pile and 2 represents the second foundation pile. Your program should move the appropriate cards according to the chosen source pile and destination pile. For example, if the source pile is 0 and the destination pile is also 0, a card is moved from the **rear** of the stock pile to the **front** of the stock pile. Your program should show the contents in the stock and foundation piles as follows:

```
Enter a source pile: 0
Enter a destination pile: 0
S: 3 * * * * *
1:
2:
```

If the user enters a number which is outside the valid range (0, 1, 2), your program should display an error message and prompt the user to enter a number again. For example, -1, 5, 8, -4 are outside the valid range, so the program displays an error message and prompts the user to enter another number.

```
S: 1 * * * * *
Enter a source pile: -1
Invalid pile number!
Enter a source pile: 5
Invalid pile number!
Enter a source pile: 0
Enter a destination pile: 8
Invalid pile number!
Enter a destination pile: -4
Invalid pile number!
Enter a destination pile: 1
S: 3 * * * *
1: [1]
2:
```

Your program should display an appropriate error message for each invalid move. For example, it is illegal to move the two cards "[1, 0]" from the first foundation pile to the second foundation pile as 3 is not equal to 1 + 1.

```
S: 2
1: [1, 0]
2: [5, 4, 3]
Enter a source pile: 1
Enter a destination pile: 2
ERROR: Invalid move!
```

Also, it is invalid to move a card from the stock pile if the stock pile is empty. For example:

```
S:
1: [1, 0]
2: [5, 4, 3, 2]
Enter a source pile: 0
Enter a destination pile: 1
ERROR: The stock pile is empty!
```

And it is invalid to move a card from the source foundation pile to the destination foundation pile if the source foundation pile is empty. For example:

```
S: 5 * * *
1: [1, 0]
2:
Enter a source pile: 2
Enter a destination pile: 1
ERROR: The foundation pile is empty!
```

Users should only be able to move cards from the stock pile to one of the foundation piles. It is invalid to move cards from a foundation pile to the stock pile. For example:

```
S:
1: [1, 0]
2: [5, 4, 3, 2]
Enter a source pile: 1
Enter a destination pile: 0
ERROR: Invalid move!
```

Your program should repeatedly prompt the user to enter two integers until all numbers are inserted into one of the foundation piles. If there are any invalid moves, the program should be able to continue to run until the end. The program prints the final congratulatory message and terminates when the game is over. The message is shown as below.

```
S: 0
1:
```

```
2: [5, 4, 3, 2, 1]
Enter a source pile: 0
Enter a destination pile: 2
S:
1:
2: [5, 4, 3, 2, 1, 0]
Congratulations!
```

The **`print_banner()`** function

- This function takes no parameter and prints the banner as shown above.

The **`print_all_piles(stock, piles)`** function

- This function takes the stock pile and two foundation piles as parameters and prints the three piles by calling the appropriate methods in the **`Stock`** and **`Foundation`** classes. For example:

```
S: 4 * *
1: [3, 2]
2: [5]
```

The **`get_pile_number(prompt)`** function

- This function displays the parameter prompt message and gets the user to enter a valid number. If the user enters a number which is outside the valid range (0, 1, 2), the function should display an error message and prompt the user to enter a number again until a valid number is entered.
- Then this function returns the entered number.

The **`game_over(stock, piles, number_of_cards)`** function

- This function returns True if the game is over. The game is over when:
  - the stock pile is empty, and
  - one of the foundation piles contains all cards and numbers are in order.
- returns False otherwise.

SUBMISSION DETAILS

Submit the following files:

- **`Foundation.py`**
- **`Stock.py`**
- **`YourUsernameGame.py`**

Your main program file must be named "`YourUsernameGame.py`", e.g. "`abcd001Game.py`", and must include a docstring at the top of the file containing your name, username, ID number and a description of the program. Complete the FIVE questions in CodeRunner and submit all of the above three files to the Assignment Dropbox: (**https://adb.auckland.ac.nz/)** at any time from the first submission date up until the final due date. You will receive an electronic receipt.

You may make more than one submission, but note that every submission that you make replaces your previous submission. Only your very latest submission will be marked.

Sample1

This section displays two example outputs using the completed program. Your program must execute in the way described and the output should have exactly the same format as the output in the examples shown (the user input is shown below in bold red). Note: The following samples are generated using **`random.seed(30)`** and where the number of cards is **`6`**.

Sample1: with valid moves only

```
CS130 Assignment - Simplified Solitaire
=======================================
S: 1 * * * * *
Enter a source pile: 0
Enter a destination pile: 0
S: 3 * * * * *
1:
2:
Enter a source pile: 0
Enter a destination pile: 1
S: 5 * * * *
```

```
1: [3]
2:
Enter a source pile: 0
Enter a destination pile: 2
S: 0 * * *
1: [3]
2: [5]
Enter a source pile: 0
Enter a destination pile: 0
S: 2 * * *
1: [3]
2: [5]
Enter a source pile: 0
Enter a destination pile: 1
S: 4 * *
1: [3, 2]
2: [5]
Enter a source pile: 0
Enter a destination pile: 2
S: 1 *
1: [3, 2]
2: [5, 4]
Enter a source pile: 1
Enter a destination pile: 2
S: 1 *
1:
2: [5, 4, 3, 2]
Enter a source pile: 0
Enter a destination pile: 2
S: 0
1:
2: [5, 4, 3, 2, 1]
Enter a source pile: 0
Enter a destination pile: 2
S:
1:
2: [5, 4, 3, 2, 1, 0]
Congratulations!
```

Sample2: with invalid moves

```
CS130 Assignment - Simplified Klondike Solitaire
================================================
S: 1 * * * * *
Enter a source pile: 0
Enter a destination pile: 0
S: 3 * * * * *
1:
2:
Enter a source pile: 0
Enter a destination pile: 0
S: 5 * * * * *
1:
2:
Enter a source pile: 0
Enter a destination pile: 1
S: 0 * * * *
1: [5]
2:
Enter a source pile: 0
Enter a destination pile: 0
S: 2 * * * *
1: [5]
2:
Enter a source pile: 0
Enter a destination pile: 2
S: 4 * * *
1: [5]
2: [2]
Enter a source pile: 0
Enter a destination pile: 2
```

```
ERROR: Invalid move!
Enter a source pile: 0
Enter a destination pile: 1
S: 1 * *
1: [5, 4]
2: [2]
Enter a source pile: 1
Enter a destination pile: 2
ERROR: Invalid move!
Enter a source pile: 2
Enter a destination pile: 0
ERROR: Invalid move!
Enter a source pile: 0
Enter a destination pile: 2
S: 3 *
1: [5, 4]
2: [2, 1]
Enter a source pile: 0
Enter a destination pile: 1
S: 0
1: [5, 4, 3]
2: [2, 1]
Enter a source pile: 0
Enter a destination pile: 2
S:
1: [5, 4, 3]
2: [2, 1, 0]
Enter a source pile: 0
Enter a destination pile: 2
ERROR: The stock pile is empty!
Enter a source pile: 0
Enter a destination pile: 1
ERROR: The stock pile is empty!
Enter a source pile: 1
Enter a destination pile: 2
ERROR: Invalid move!
Enter a source pile: 2
Enter a destination pile: 1
S:
1: [5, 4, 3, 2, 1, 0]
2:
Congratulations!
```

Marking Criteria

1	1 mark	Include a docstring with your name, Student ID, username and a program description at the top of your file.
2	1 mark	The code is self-documenting (good, meaningful variable names, etc).
3	1 mark	The program is easy to read (e.g. spaces between operators and operands, etc).
4	3 marks	The Stock class is implemented correctly (CodeRunner questions)
5	2 marks	The Foundation class is implemented correctly (CodeRunner questions)
6	1 mark	The program runs correctly as specified.
7	5 marks	The program displays an error message for each incorrect move from the stock pile.
8	4 marks	The program displays an error message for each incorrect move from the foundation pile.

Style Conventions

- Descriptive variable names that follow Python naming conventions.
- None of your methods/functions should be longer than 30 lines. You may define helper methods/functions if needed.
- A space between all operators and operands and after commas.

ACADEMIC INTEGRITY

The purpose of this assignment is to help you develop a working understanding of some of the concepts you are taught in the lectures and labs. We expect that you will want to use this opportunity to be able to answer the corresponding questions in the tests and exam. We expect that the work done on this assignment will be your own work. We expect that you will think carefully about any problems you come across, and try to solve them yourself before you ask anyone for help. The following sources of help are not acceptable:

- Getting another student, or other third party to instruct you on how to design the functions or have them write code for you.

- Taking or obtaining an electronic copy of someone else's work, or part thereof.
- Giving a copy of your work, or part thereof, to someone else.
- Using code from online sources.

The School of Computer Science uses copy detection tools on all submissions. Submissions found to share code with those of other people will be detected and disciplinary action will be taken. To ensure that you are not unfairly accused of cheating:

- Always do individual assignments by yourself.
- Never give any other person your code or sample solutions in your possession.
- Never put your code in a public place (e.g., Piazza, forum, your web site).
- Never leave your computer unattended. You are responsible for the security of your account.
- Ensure you always remove your USB flash drive from the computer before you log off, and keep it safe.