Lab Exercise #12

Assignment Overview

You will work with a partner on this exercise during your lab session. Two people should work at one computer. Occasionally switch the person who is typing. Talk to each other about what you are doing and why so that both of you understand each step.

Part A: The Card and Deck Classes

1. Consider the Python statements shown below (from the file named "lab12.parta.py") and the partial output they produce. Predict what the remaining output will be, then test your predictions.

```
print( "===== initial deck =====" )
my_deck.display()

===== initial deck =====

A♣  2♣  3♣  4♣  5♣  6♣  7♣  8♣  9♣  10♣ J♣  Q♣  K♣
A♦  2♦  3♦  4♦  5♦  6♦  7♦  8♦  9♦  10♦ J♦  Q♦  K♦
A♥  2♥  3♥  4♥  5♥  6♥  7♥  8♥  9♥  10♥ J♥  Q♥  K♥
A♠  2♠  3♠  4♠  5♠  6♠  7♠  8♠  9♠  10♠ J♠  Q♠  K♠

my_deck.shuffle()
print( "===== shuffled deck =====" )
my_deck.display()

===== shuffled deck =====

10♥ K♠  J♦  3♥  8♠  J♣  5♣  Q♠  4♠  9♣  6♥  10♣ 5♦
4♥  2♣  2♦  6♣  K♥  K♦  10♠ 9♥  6♦  3♠  A♠  9♦  8♥
A♥  4♣  7♠  3♦  5♠  10♦ Q♣  8♦  7♥  K♣  8♣  Q♥  7♣
2♥  J♥  9♠  6♠  4♦  3♣  5♥  2♠  7♦  A♦  A♣  J♠  Q♦

card1 = my_deck.deal()
print( "First card dealt from the deck:", card1 )
print()
print( "Card suit:", card1.suit() )
print( "Card rank:", card1.rank() )
print( "Card value:", card1.value() )
print()
print( "Deck empty?", my_deck.is_empty() )
print( "Number of cards left in deck:", len(my_deck) )
print()

First card dealt from the deck: Q♦

Card suit: ??
Card rank: ??
Card value: ??

Deck empty? ??
Number of cards left in deck: ??
```

```
card2 = my_deck.deal()
print( "Second card dealt from the deck:", card2 )
print()
print( "Card suit:", card2.suit() )
print( "Card rank:", card2.rank() )
print( "Card value:", card2.value() )
print()
print( "Deck empty?", my_deck.is_empty() )
print( "Number of cards left in deck:", len(my_deck) )
print()

Second card dealt from the deck: ??

Card suit: ??
Card rank: ??
Card value: ??

Deck empty? ??
Number of cards left in deck: ??

if card1.suit() == card2.suit():
    print( card1, "same suit as", card2 )
else:
    print( card1, "and", card2, "are from different suits" )

?????

if card1.rank() == card2.rank():
    print( card1, "and", card2, "of equal rank" )
elif card1.rank() > card2.rank():
    print( card1, "of higher rank than", card2 )
else:
    print( card2, "of higher rank than", card1 )

?????

if card1.value() == card2.value():
    print( card1, "and", card2, "of equal value" )
elif card1.value() > card2.value():
    print( card1, "of higher value than", card2 )
else:
    print( card2, "of higher value than", card1 )

?????
```

2. Examine the Python statements in the file named “cards.py”, then answer the following questions.

   a. What are the rank and the value of an Ace?

   b. What are the rank and the value of a King?

   c. What is the difference between the rank of a card and the value of a card?

Part B: Using the Class Methods

1. Examine the Python statements in the file named "lab12.partb.py", then extend that program to do the following tasks:

a. Display each players' hand after the first card has been played from each hand.

b. Display the *second* card dealt to each player and compare them.

c. Display each players' hand after the second card has been played from each hand.

d. Display the *last* card dealt to each player and compare them.

2. Revise the program to use a different integer number as the argument in the invocation of function "random.seed". Run the program several times and observe the results.

3. Revise the program to eliminate the invocation of function "random.seed". Run the program several times and observe the results. Note: if function "random.seed" is not invoked, then the current system time is used to initialize the random number generator.

★ **Demonstrate your completed program to your TA. On-line students should submit the completed program (named "lab12.partb.py") for grading via the CSE handin system.**

Part C: Programming with the Class Methods

Consider the program in the file named "lab12.partc.py", which imports the module containing the Card and Deck classes. Extend that program to play the card game named "War". Note: you may not modify the contents of "cards.py" (the Card and Deck classes).

Game Rules (Simple Version)

War is a two-player game which uses a standard deck of 52 cards. Suits are ignored in the game, and the cards are ordered as follows: Ace King Queen Jack 10 9 8 7 6 5 4 3 2 (Aces are the highest cards).

The cards are distributed to the two players one at a time (alternating), until both players have a stack of 26 cards.

Each player turns over the top card on his stack (a battle). Whoever played the card with the highest rank wins the battle and adds both cards to the bottom of his stack. In the simple version, if both cards have the same rank, each player simply returns his card to the bottom of his own stack (the battle was a stalemate). Play continues until one player has all 52 cards in his stack and wins the game.

Programming Notes

1. After dealing out the cards, your program will repeatedly process one battle. After each battle, your program will ask the user if he wishes to continue or not (make sure that simply touching the Enter key is the default and means "continue").

2. For each battle, your program will display the card played by each player, announce the result of the battle, and display each player's stack of cards. If one player wins the game as the result of the battle, your program will announce the winner and halt.

3. If the user chooses to quit the game after a battle, the player with the most cards will be declared the winner.

4. In the game of War, Aces are the highest-ranked cards. However, in the Card class, Aces have the lowest rank. Your program must account for this difference.

5. Your program must model each player's stack of cards. One approach would be to use a list of cards, where slot 0 of the list is viewed as the top of the stack. The list method "append" could be used to add a card to the bottom of the stack.

6. Your program will be subdivided into functions which will pass parameters and return values to communicate between the functions.

7. When played with 52 cards, the game can take a long time to complete. To test your logic in a less time-consuming way, have your program deal out a much smaller number of cards (perhaps five to each player).

★ **Demonstrate your completed program to your TA. On-line students should submit the completed program (named "lab12.partc.py") for grading via the CSE handin system.**

Game Rules (Full Version)

Optional: if you have the time and interest, revise your program to handle the full version of the game rules.

The rules for the full version of War are the same as above, with one exception. Instead of a stalemate when both players turn over a card with the same rank, the battle continues. Each player turns over two more cards, and the second card of each pair is used to determine the result of the battle. If one player's card outranks the other player's card, the battle is over and the winner adds all six cards to the bottom of his stack. If the two cards have the same rank, the battle continues (as above).

The game rules for the full version do not specify what happens if one player runs out of cards during a protracted battle. For example, assume both players turn over a King. Each player is supposed to put down two more cards, but one of the players only has one card in his stack.

Use the following procedure to resolve this situation. If one player runs out of cards without being able to complete a protracted battle, the other player wins. If both players run out of cards simultaneously, the game is a draw (neither player wins).