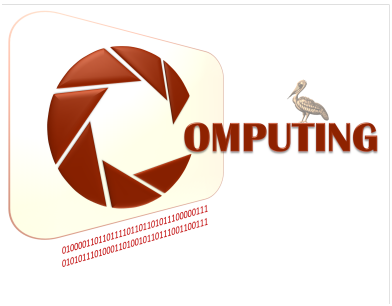

COMP1127 – Introduction to Computing II
2017-2018 Semester I
Assignment

Introduction

There is a game of playing cards that is popular across the Caribbean. In Trinidad and Tobago, and Barbados it is known as Suck the Well, in Jamaica it is known as Strip Me. For your assignment you will create a version of this game using the problem solving approaches, programming tools, Python and the IDLE integrated development environment. So let us begin.

About the game

All that is required is a full deck of playing cards, i.e. a pack of 52 playing cards. The deck of cards is divided among the players. There must be at least 2 players for this game. An even number of players results in an even distribution of cards, an odd number of players results in one player having 1 card greater than the others. The version that you will be developing will be a 2 player game, you versus the computer.

For a 2 player game, each player will therefore be dealt 26 cards. Like many other games, the deck of cards should be shuffled before the start of the game. Each player holds their hand of cards face down, players do not see what card they will play next. A play occurs when a player removes the top card of their hand and places it face up at the top of a discard pile. Players can only ever see the top card of the discard pile.

There are 2 groups of cards, pay cards and ordinary cards. Pay cards are Jacks, Queens, Kings, Aces. All other cards are ordinary cards. Suits do not matter. Play continues alternately until a pay card appears on the discard pile. The opponent of the person who played the pay card must pay for it by playing several times in succession. The payment rates are as follows,

- 4 ordinary cards for an ace
- 3 ordinary cards for a king
- 2 ordinary cards for a queen
- 1 ordinary card for a jack.

When the payment is completed the person who played the pay card takes the entire discard pile and adds it face down to the bottom of their hand. If a pay card appears while a player is making a payment, the previous pay card is canceled and the opponent now has to pay for this new pay card. This continues until a payment is completed by one of the players. The player who first runs out of cards loses.

What you should and will have

Your solution should be developed using Python version 3.6.0. This is the version you should already be familiar with. If not, download it. The assignment and its solution were developed using this version. You also should have been writing your own solutions for the duration of the courses (COMP1126 and COMP1127). If not, you will have difficulty in completing this assignment. This document will have examples of how the various elements are expected to work. The examples should help you verify that the code you have written does what is expected. You will be provided with a module called **`cards`**. This module provides some functions that you may find useful in developing your solution. The functions , their type signatures and brief description of their services are below.

`new_Deck: void → deck`	Creates a new, full deck of cards
`shuffle: round,deck → deck`	Shuffles a deck of cards, a number of times specified by 'round'
`deal: deck,# of cards,# of players → hands,deck`	Deal '# of cards' to each of '# of players' from 'deck'. Returns a number of hands equal to the number of players and the remaining deck
`getSuitIcon: card → code`	Retrieve the code for the icon for a suit

A card is represented as a tuple of two characters, a suit and a face value. Suits (hearts ♥, clubs ♣, spades ♠ and diamonds ♦) are represented as "H","C","S","D"; face values as "2","3","4","5","6","7","8","9","10","J","Q","K","A". For example, the two of spades is represented as ('S', '2') and the queen of hearts as ('H','Q'). A deck of cards is represented as a list of cards. To use the `cards` module, save it to the same folder that you will be saving your Strip Me solution. In your script, you will then import the `cards` module and use it like any other (for example remember how the `math` module has been used). If saving to the same folder proves unsuccessful, save the module to one of IDLE's path directories. You can find these in IDLE by selecting '`File`' then '`Path Browser`'.

What you need to do

We will build the game by developing the smaller components/aspects of the game and then putting these components together. This is the same method that we have been imparting to you as an approach to solving complex problems.

1. Two core aspects of the game are the players' hands of cards and the discard pile. From the description of the game play you may have already realised the hands can be implemented as a queue, the discard pile as a stack. Therefore you need to create the following abstract data types,

Queue ADT

`new_Queue: void → queue`	Create a new, empty queue
`is_Queue: queue → boolean`	Is the argument a queue that conforms with the representation?
`queue_Contents: queue → list`	Return the contents of the queue argument
`empty_Queue: queue → boolean`	Is the queue argument empty?
`queue_Front: queue → element`	Return the element at the front of the queue argument
`enqueue: queue, element → queue`	Add an element to the back of the queue argument
`dequeue: queue → queue`	Delete the element at the front of the queue argument

Stack ADT

`new_Stack: void → stack`	Create a new, empty stack
`is_Stack: stack → boolean`	Is the argument a stack that conforms with the representation?
`stack_Contents: stack → list`	Return the contents of the stack argument
`empty_Stack: stack → boolean`	Is the stack argument empty?
`stack_Top: stack → element`	Return the element at the top of the stack argument
`push: stack, element → stack`	Add an element to the top of the stack argument
`pop: stack → stack`	Delete the element at the top of the stack argument

Now that you have the basic blocks of a solution i.e. the `cards` module, queue and stack ADTs, we can begin developing our game.

2. It is always good to have a greeting. For example,

```
*********************************************************
*        WELCOME to Strip Me                            *
*                                                       *
* Rules of the game: Check the docs                     *
*                                                       *
* Game Play:                                            *
* - player 0: the computer                              *
* - player 1: you                                       *
* - enter: play the top card and place on discard pile  *
* - q: quit i.e. stop playing the game                  *
*                                                       *
*              Enjoy!                                   *
*********************************************************
```

You can use a series of print statements within a procedure to accomplish this. Name this procedure `showGreeting`. The procedure does not accept any parameters.

3. Include a small dictionary called `payCards`. There must be 4 elements. Each element has the format face:pay rate. For example `'K':3` stores the information that a king's pay rate is 3

4. Now that we have `payCards`, write a function that determines whether a card is a pay card. Name the function `isPayCard`. For example,

```
>>> isPayCard(('H',4))
False
>>> isPayCard(('S','A'))
True
```

5. Write a function called `getCardRate` which accepts a card as its only argument. If the card argument is a pay card, it returns its pay rate, otherwise it returns 0. For example,

```
>>> getCardRate(('S','A'))
4
>>> getCardRate(('H',4))
0
```

With the above, we can now determine whether a card is a pay card or an ordinary card. We now need to be able to distribute cards to players and get them ready to play.

6. We need to be able to deal cards to a player. A player's hand of cards will be implemented using a queue, as was mentioned earlier. Write a function called `fillHand` that accepts a queue and a list of cards as its arguments. It returns the queue argument with the cards from the list of cards added to the queue. The entire list of cards is not en-queued as one element, its elements (cards) are en-queued one at a time. In the example that follows a player's hand (a queue) is created that will be used to show the function of `fillHand`. The module `adt` in the example is not provided to you, you are writing your own. For example,

```
>>> hand=adt.new_Queue()
>>> hand
('Queue', [])
>>>hand=fillHand(hand,[('C',10),('D','Q'),('H',5),('S',8)])
>>> hand
('Queue', [('C', 10), ('D', 'Q'), ('H', 5), ('S', 8)])
```

7. Now, we can have a player accept cards when we deal a hands of cards to them. After this happens the game can actually begin. Write a function called `prepPlayers`. This function does not accept any parameters. The function creates a new deck of cards, shuffles it, and deals 26 cards to each of 2 players. It then fills the players' hands of cards, using `fillHand`, and returns the filled hands as a tuple. For example,

```
>>> playersHands=prepPlayers()
>>> playersHands
[('Queue', [('S', '10'), ('C', '8'), ('S', 'J'), ('S', 'A'),
('S', '2'), ('S', '7'), ('C', '6'), ('C', '4'), ('D', 'Q'),
('D', '3'), ('S', '3'), ('D', '4'), ('H', '7'), ('D', '6'),
('H', '2'), ('D', 'K'), ('H', 'J'), ('C', 'Q'), ('H', '8'),
('H', '5'), ('S', '5'), ('D', '8'), ('D', '10'), ('H', '9'),
('C', '3'), ('S', '4')]), ('Queue', [('H', '10'), ('H', '4'),
('D', '9'), ('C', '5'), ('D', '2'), ('H', '6'), ('D', '7'),
('S', '9'), ('S', 'Q'), ('S', '6'), ('S', 'K'), ('H', 'Q'),
('C', 'J'), ('D', 'A'), ('C', 'A'), ('H', 'K'), ('C', '10'),
('C', '2'), ('C', '7'), ('S', '8'), ('D', '5'), ('C', '9'),
('C', 'K'), ('H', 'A'), ('H', '3'), ('D', 'J')])]
```

8. The players have their cards, so they can begin playing the game. Essential to game play is a player's ability to take the top card of their hand and place it on the discard pile. Write a function called `placeCard` that performs this service.

It accepts a player's hand of cards and the discard pile as arguments. It then adds the card at the top of that player's hand (remember this is a queue) and places it at the top of the discard pile (remember this is a stack). It returns the updated player's hand of cards and the updated discard pile. For example (using the `playersHands` in part 7 above),

```
>>> placeCard(playersHands[0],adt.new_Stack())
(('Queue', [('C', '8'), ('S', 'J'), ('S', 'A'), ('S', '2'),
('S', '7'), ('C', '6'), ('C', '4'), ('D', 'Q'), ('D', '3'),
('S', '3'), ('D', '4'), ('H', '7'), ('D', '6'), ('H', '2'),
('D', 'K'), ('H', 'J'), ('C', 'Q'), ('H', '8'), ('H', '5'),
('S', '5'), ('D', '8'), ('D', '10'), ('H', '9'), ('C', '3'),
('S', '4')]), ('Stack', [('S', '10')]))
```

9. With the ability to place a card, game play can now begin. Write a function called `playCard`. This function takes 3 arguments: the current player, their hand of cards, the discard pile. It places the card using `placeCard`, displays what card was played, returns the updated player's hand of cards and the updated discard pile. Do not display a card using its internal representation i.e. as a tuple. Display it with its face value and its suit icon. Continuing the previous example and using `playersHands`,

```
>>> playCard(0,playersHands,adt.new_Stack())

Player  0 ,
played the  8 ♣
(('Queue', [('S', 'J'), ('S', 'A'), ('S', '2'), ('S', '7'),
('C', '6'), ('C', '4'), ('D', 'Q'), ('D', '3'), ('S', '3'),
('D', '4'), ('H', '7'), ('D', '6'), ('H', '2'), ('D', 'K'),
('H', 'J'), ('C', 'Q'), ('H', '8'), ('H', '5'), ('S', '5'),
('D', '8'), ('D', '10'), ('H', '9'), ('C', '3'), ('S', '4')]),
('Stack', [('C', '8')]))
```

10. We now need to be able to have a player accept payment. Write a function called `take Payment`. It accepts a player's hand of cards and the discard pile as arguments. It adds all of the cards, one at a time, from the discard pile to the bottom of the player's hand of cards. For example,

```
>>> playerHand
('Queue', [('C', '4'), ('D', 'Q'), ('D', '3'), ('S', '3'),
('D', '4'), ('H', '7'), ('D', '6'), ('H', '2'), ('D',  'K'),
('H', 'J'), ('C', 'Q'), ('H', '8'), ('H', '5'), ('S', '5'),
('D', '8'), ('D', '10'), ('H', '9'), ('C', '3'), ('S', '4')])
```

```
>>> discardPile
('Stack', [('S', 'A'), ('S', '2'), ('S', '7'), ('C', '6')])
>>> takePayment(playerHand,discardPile)
(('Queue', [('C', '4'), ('D', 'Q'), ('D', '3'), ('S', '3'),
('D', '4'), ('H', '7'), ('D', '6'), ('H', '2'), ('D',  'K'),
('H', 'J'), ('C', 'Q'), ('H', '8'), ('H', '5'), ('S', '5'),
('D', '8'), ('D', '10'), ('H', '9'), ('C', '3'), ('S', '4'),
('C', '6'), ('S', '7'), ('S', '2'), ('S', 'A')]), ('Stack', []))
```

11. We can now simulate the full game play. Write the main procedure called `strip_me`. Take a look at a bit of game play on the following page. `strip_me` should do the following,

    - show the greeting
    - initialise variables that will be used, for example players, their piles, the discard pile
    - continue playing a game until either the user quits or a player loses. This involves,
        - prompting the user to play or quit
        - checking what card has been played to determine its rate
        - keeping track of which player's turn it is to play
        - if a pay card, having the player pay and the other take the payment
        - displaying appropriate messages during game play and when the game ends

12. Finally, include a call that begins a game by invoking `strip_me`.

```
*********************************************************
*          WELCOME to Strip Me                          *
*                                                       *
* Rules of the game: Check the docs                     *
*                                                       *
* Game Play:                                            *
* - player 0: the computer                              *
* - player 1: you                                       *
* - enter: play the top card and place on discard pile  *
* - q: quit i.e. stop playing the game                  *
*                                                       *
*                    Enjoy!                             *
*                                                       *
*********************************************************
play(Enter); quit(q,then enter)

Player  1 ,
played the  2 ♣

Player  0 ,
played the  Q ♦

play(Enter); quit(q,then enter)

Player  1 ,
played the  A ♦

Player  0 ,
played the  Q ♣

play(Enter); quit(q,then enter)

Player  1 ,
played the  4 ♥

Player  1 ,
played the  9 ♠

 The full payment was made. Player  0  claimed the discard pile

Player  0 ,
played the  6 ♠

play(Enter); quit(q,then enter)q
You quit the game before it ended, so there's no result. Bye!
```

That's it. Hope you had fun

This assignment is worth 15% of your coursework mark. It is due on November 24th 2017, at 11pm. Post your submission to the course container on OurVLE. Look for the container named "Assignment Submissions".

You are required to work in pairs, therefore ensure that your code has BOTH members' ID numbers included as a comment. Only one member of the pair is to submit. Each person may submit as a member of one (1) programming pair only. No late submissions will be accepted.

Name your file according to the following convention,

IF the ID numbers of the students are 620000001 and 620000002
THEN the file name should be 620000001_620000002.py

Do NOT deviate from this naming convention!