Spring, 2015 CIT 590

Programming Languages and Techniques

Homework 3

Due as per deadline on canvas

This homework deals with the following topics

* lists

* being creative in creating a game strategy (aka having fun)

General Idea

This assignment is mainly intended to get you to practice list modifications

We will be implementing the old game Racko which is a game that involves rearranging your hand of cards in order to have an increasing sequence.

While Racko is typically played with 2 to 4 players, we will keep this simple and just play the user versus the computer. The user's moves are decided by the user by asking for input, the computer's moves are decided by you the programmers! What that means is that there is NO right answer for the section that asks you to program a strategy for the computer. All we want you to do is come up with some reasonable enough strategy that ensures that a human user does not consistently beat the computer. So unlike your previous assignments, this one has a creative component to it as well.

Rules of the game

A Racko deck is composed of 60 cards, each numbered 1 to 60.

The objective is to be the first player to arrange all of the cards in your rack from lowest to highest. Each players rack has 10 slots i.e. 10 cards will ultimately have to be in ascending order for the player to win the game.

To start the game, shuffle the deck, both the user and the computer pick a card from the deck. Let the computer pick first. The person who gets to play first is the person who has the higher number.

The cards then get shuffled again and both the user and the computer gets dealt 10 cards. As a player receives each card, he must place it in the highest available slot in his rack, starting at slot 10, without rearranging any of them. The goal of each game is to create a sequence of numbers in ascending order, starting at slot 1.

The top card of the deck is turned over to start the discard pile. A player takes a turn by taking the top card from either the deck or the discard pile, then discarding one from his rack and inserting the new card in its place. If the player draws a card from the deck, he may immediately discard it; if he takes the top discard, though, he must put it into his rack.

The first player to get his 10 cards in ascending order calls "Rack-O!" and wins the hand.

While I do not have a link to an online Racko game, there is a very similar game (with little Vikings!) over here tower blaster

We also have the actual game with us (assuming I did not forget to bring it with me to recitation) and are more than happy to pass it around in class for you to get a feel for what the game is like.

Finally here is just a random picture of the game to remind yourself what it looks like (you can also stop by my office to take another look).

RACKO
The Classic Rack 'Em & Score Card Game
Great Family Fun for over Fifty Years!
8+

The actual program

Once again we provide you with stubs of functions. But this time around there is less guidance as to what exactly you need to put in them.

Also, in this rare instance, we will be using 2 global variables. One called **deck** and one called **discard**. This is not the best design, but it is ok for now. We will redesign this once we cover things like classes.

Here are the functions that need to be written.

We are expecting to see these functions with these names and the same number of arguments

Note that we will make heavy use of lists in the assignment. Since a rack looks more like a vertically oriented structure, we need to have some convention. Our convention is that

```
lst = [3, 17, 11, 30, 33, 38, 49, 46, 25, 53]
```

is actually the following rack from slots 10 down to 1

```
3
17
11
30
33
38
49
46
25
53
```

So yes, this particular rack is a long way from a victory.

I know this is a somewhat awkward way to representing the data, but I'm deliberately asking you to do it this way in order for you to do more list exercises.

The deck and discard pile are also going to be represented as lists. Since we want to simulate the idea of a stack of cards we will make heavy use of the functions list.append and list.pop. The only cards you have access to during the game are the last cards in these lists, lst[-1]. This is to simulate the stack of cards that you have. The list function called pop gives you the last element in a list and kicks it out of the list.

The list of functions begins here

I have used the terms hand and rack interchangeably. What is meant by hand is basically just the arrangement/set of cards on the rack.

- **shuffle()** - This function shuffles the deck or the discard pile. To decide which one it needs to shuffle, it checks the length of the deck. It shuffles the deck to start the game. Also used to shuffle the discard pile if we ever run out of cards in the deck (length of deck being 0).

- **check_racko(rack)** - given a rack (this will be either the user's or the computer's) determine if Racko has been achieved. Remember that Racko means that the cards are in ascending order.

- **deal_card()** - get the top card from the deck. Used at the start of game play and again during each player's turn if they decide to not take the top card from the discard pile.

- **deal_initial_hands()** - start the game off by dealing two hands of 10 cards each. This function returns two lists - one representing the user's hand and the other representing the computer's hand.

  Make sure that you follow normal card game conventions of dealing. So programatically you have to deal one card to the user, one to the computer, one to the user, one to the computer and so on.

  Also the computer is always the first person that gets dealt to. Just keeping it simple.

  Remember that the rules of our version of Racko will be that you have to place your cards in the order of top most slot of the rack first, then the next slot and so on.

  In the example above, this would mean that someone was dealt the cards 3, 17, 11, 30, 33, ... in that order.

  Remember the discussion we had in the lecture(it is also there in the book) about how to return two things from a function. Use that idea over here.

- **does_user_begin()** - simulate a coin toss by using the random library. If it is 'heads', the user goes first. If 'tails', the computer goes first.

- **print_top_to_bottom(rack)** - given a rack(represented as a list) print it out from top to bottom in a manner that looks more akin to the game (more stack like than list like). See the example above for the exact specification. Please stick to that specification in terms of the representation of the rack.

- **find_and_replace(newCard, cardToBeReplaced, hand)** - find the cardToBeReplaced (represented by a number) in the hand and replace it with newCard. The replaced card then gets put on top of the discard. Check and make sure that the cardToBeReplaced is truly a card in the hand. If the user accidentally typed a card number incorrectly, just politely remind her to try again and leave the hand unchanged.

- **add_card_to_discard(card)** - add the card(represented as just an integer) to the top of the discard pile.

- **computer_play(hand)** - This function is where you can write the computer's strategy down. It is also the function where we are giving you very little guidance in terms of actual code. All we want you to do is take in one input - the computer's hand and use the discard and the deck(remember those are global variables) to figure out what the computer should be doing.

You are supposed to be somewhat creative here, but I do want your code to be deterministic. That is, given a particular rack and a particular card (either from the discard pile or as the top card of the deck), you either always take the card or always reject it.

Here are some potential decisions the computer has to make

1. Given the computer's current hand, do you want to take the top card on the discard or do you want to take the top card from the deck and see if that card is useful to you.
2. How do you evaluate the usefulness of a card and which position should go into.
3. There might be some simple rules you can give the computer. For instance, it is disastrous to put something like a 5 in the top slot. You want big numbers over there.

You are allowed to do pretty much anything in this function except make a random decision or make a decision that is obviously incorrect. For instance, making your bottom card a 60 is a recipe for disaster. Also, the computer CANNOT CHEAT. What does that mean? The computer cannot peek at the top card of the deck and then make a decision of going to the discard. It's decision making should be something that a human with a good amount of processing power should be able to make as well.

This function has to return the new hand for the computer.

- main() - a function that puts it all together. You play until someone gets Racko.

  We have a basic structure for main() in the next page. We deliberately have not written real code. The intent is that you fill in the required pieces of code once you understand our comments.

  You do not have to adhere to this structure of main(). You can use your own design as long as you stick to the rules of the game.

```
def main():
    global deck
    global discard
    shuffle()
    #deal a card to the computer and a card to the user
    #repeat until both have 10 cards
    userStarts = does_user_begin()
    print_top_to_bottom(human_hand)
    #reveal one card to begin the discard pile
    while neither the computer nor the user has racko:
        computer_hand = computer_play(computer_hand)
        #ask the user if they want this card
        #print the user's hand
        if user chooses this card:
            #ask the user for the number of the card they want to kick out
            #modify the user's hand and the discard pile
            #print the user's hand
        elif choice == 'n':
            card = deck.pop()
            print 'The card you get from the deck is ' + str(card)
            #ask the user if they want this
            secondChoice = raw_input('do you want to keep it?\n')
            if secondChoice == 'y':
               #modify user's hand, the discard pile and then print user's hand
            else:
                discard.append(card)
                #print the user's hand
        #check and make sure there are still some cards in the deck
        #else reshuffle the discard and restart.
```

Evaluation

The primary goal of this assignment is to get you to feel familiar with lists and to have some level of fun while creating a game.

While we want you to spend time on coming up with some kind of strategy for the computer, that is NOT the primary part of the assignment. Come up with something reasonable. Any reasonable strategy will have you doing some fun things with lists. If the user always does absolutely nothing at all, that is, they reject the discard card and they reject the top card from the deck and move it onto the discard, then we want the computer to beat the user. Your computer should have enough intelligence to beat the 'stupid and lazy' user.

Also, remember the user has no idea what you are doing internally in your functions and does not want to be shown a large amount of print statements. At any given point in the game, they should know only 3 things - their entire hand printed in the rack form, the top card of the discard and if they choose to dive into the deck they get to know the top card of the deck.

Evaluation criteria

- 8 points for getting the game to work.
- 5 points for style - variable names, function names, modularity.
- 4 points for getting the specifications correct (passing all the TAs tests).
- 3 points for strategy

what to submit

Submit one single file called Racko.py. Only 1 of you should submit.

Also please please stick to the given function names.

Please put the following at the end. Now that you have seen it a few times, we will deduct a point if you do not have this at the end.

```
if __name__ == '__main__':
    main()
```