CS 1133, LAB 4: BLACKJACK

http://www.cs.cornell.edu/courses/cs1133/2013fa/labs/lab04.pdf

First Name: _____ **Last Name**: _____ **NetID**: _____

This lab is *a lot* shorter than you might realize at first glance. You already have enough work with the assignment due this week, so we just wanted a simple assignment to give you some practice with classes.

Lab Materials. We have created several Python files for this lab. You can download all of the from the Labs section of the course web page.

http://www.cs.cornell.edu/courses/cs1133/2013fa/labs

When you are done, you should have the following three files.

- `blackjack.py` (the primary module for the lab)
- `test_blackjack.py` (a unit test for `blackjack.py`)
- `card.py` (a support module, which you will not touch)

You should create a *new* directory on your hard drive and download all of the files into that directory. Alternatively, you can get all of the files bundled in a single ZIP file called `lab04.zip` from the Labs section of the course web page.

0.1. **Getting Credit for the Lab.** Everything in this lab will either be written on this handout, or implemented in `blackjack.py`. When you are done, show both of these to your instructor. You instructor will then give you credit for the lab. You do not need to submit the paper with your answers, and you do not need to submit the computer files anywhere.

As with the previous lab, if you do not finish during the lab, you have **until the beginning of lab next week to finish it**. You should always do your best to finish during lab hours. Remember that labs are graded on effort, not correctness.

---

1. The Game of Blackjack

In this lab, you will finish a class definition for `Blackjack` that a casino could use to run multiple blackjack games simultaneously.

A player wins at blackjack by ending with a hand that has more points than the dealer's, but not more than 21 points. If someone exceeds 21 points, they are said to have "gone bust" and immediately lose. Points come from the ranks of the cards in a hand: 10 points for each face card (Jack, Queen, or King), 11 points for an ace, and the rank of the card for anything else (e.g. a 4 of anything is 4 points). In some games of blackjack, an ace can be worth either 1 or 11, whichever is better; we will ignore that rule for our implementation.

Course authors: D. Gries, L. Lee, S. Marschner, W. White

Play begins with two cards being dealt to the player and one card to the dealer. All cards in each hand are always visible to all participants. The player can chose to "hit" (get an additional card from the deck) or "stay" (turn over play to the dealer). If the player eventually stays without going bust, then the dealer draws cards until she goes bust or decides to stop.

Once you complete the lab, you can relax and play a few rounds of the game yourself. The file `blackjack.py` has script code, and so can be safely run as a script. He is a sample transcript showing off a working game:

```
[llee: lab08] python blackjack.py
Welcome to CS 1110 Blackjack.
Rules: Face cards are 10 points. Aces are 11 points.
       All other cards are at face value.

Your hand:
8 of Spades
6 of Clubs

Dealer's hand:
9 of Spades

Type h for new card, s to stop: h
You drew the 6 of Spades

Type h for new card, s to stop: s

Dealer drew the 3 of Spades
Dealer drew the 4 of Spades
Dealer drew the 8 of Hearts
Dealer went bust, you win!

The final scores were player: 20; dealer: 24
```

1.1. **The Module `card`.** The Card class is provided by the module `card`. You do not need to do anything with this module at all. You might want to check out the Card methods, but that is not necessary. The helper functions in `blackjack.py` take care of all of those details for you.

2. Completing the Blackjack class definition

You should proceed in an iterative fashion. For each step outlined below,

(1) Read the directions in this handout and the specification of the relevant methods.

(2) Look at the appropriate test cases in `test_blackjack.py`, to better understand the goal.

(3) Remove lines with the comment "implement me", and write the appropriate code.

(4) Test your code using `test_blackjack.py`. You do not need to add test cases to it.

Make sure each method passes its test cases *before* moving on to implement the next method. This is important because many of the methods here build on earlier ones.

2.1. **Fix the method headers.** There is something wrong in at least one of the headers of the methods for class `Blackjack`. You can tell by running the test script `test_blackjack.py` in the command shell.

What error do you get, and how should you fix the error? Write your answers below, and *then fix* all *method headers that require this correction.*

2.2. **Implement and test `__init__`.** Implement `__init__` so that it initializes the three instance attributes of `Blackjack`. For this part, you will probably want to make use of standard list operations. For reference, look at section 5.1 in the Python library at

`http://docs.python.org/2/tutorial/datastructures.html`

Our solution is three lines long. Write yours here:

2.3. **Enforce the preconditions for `__init__`.** Notice that the `__init__` method has preconditions for the parameter `deck`. You can break this precondition into three facts: `deck` is a list, `deck` contains only `Cards`, and `deck` has at least three elements.

At least two of these are relatively easy to enforce (All three are enforceable if you are really clever; look up the `all` function in Python). In the box below, write assert statements that would enforce at least two of these preconditions.

2.4. **Read over but don't change `_score`.** Note the leading underscore in this method. This is meant to be a private helper method for the class (and for you).

2.5. **Implement `dealerScore()` and `playerScore()`.** You should use the private helper method that we have provided. The syntax for getting this right might take a little getting used to, so write down your code for `dealerScore` here for your instructor to take a look at:

2.6. **Implement and test `playerBust()` and `dealerBust()`.** Your implementation should use `dealerScore()` and `playerScore()` as helper methods.

2.7. **Implement and test `__str__`.** . Note that this method is "higher up" in the file, just after `__init__`, as is conventional. Use `dealerScore()` and `playerScore()` as helper methods.

2.8. **Play some Blackjack!** This last part is just for fun. Run `blackjack.py` as a script:

```
python blackjack.py
```

Follow the directions on the screen. The command 'h' is for 'hit', and 's' is for stay.

Our dealer is following a common house protocol. As with most casinos, the dealer must continue to hit while her hand is under 17. Once her hand reaches 17 or more, she must stay (or go bust). See if you can use this to your advantage.

2.9. **OPTIONAL CHALLENGE.** You are done with the lab, but if you want an extra challenge, you can try this. In real blackjack, aces can count as either 1 point or 11 points, depending on what is most advantageous for the holder of the hand. The `_score` method would have to be rewritten to account for that.

What should a modifed `_score` method do. Sould it return a range of possible scores for a hand? A list of possible scores? The best possible score? How would you change your code according to this design decision?