CS 1110, LAB 9: BLACKJACK

http://www.cs.cornell.edu/courses/cs1110/2017fa/labs/lab9/

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, and later assignments will be longer. Therefore, we thought it best to give you a straight-forward lab that built up some practice with classes.

1. Lab Files

For today's lab you the following additional files:

- `lab09.py` (the primary module for the lab)
- `test09.py` (a completed unit test script to aid you)
- `card.py` (a support module, which you will not touch)

Once again 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 lab09.zip from the Labs section of the course web page.

While there are several files here, you will only modify the first file, `lab09.py`. Everything else is either a support module (providing another class) or something to help you test and debug.

Getting Credit for the Lab. Once again, you have a choice between getting credit through the online system or your instructor. The online lab is available at the web page

http://www.cs.cornell.edu/courses/cs1110/2017fa/labs/lab9/

If you choose to stick with this worksheet, keep in mind that you should write answers both on this sheet, and in the file `lab09.py`. You will need to show both of these to your instructor to get credit.

As with all previous labs, if you do not finish during the lab, you have until the **beginning of lab next week to finish it**. Over the next week, you may either (1) complete the lab online, (2) show your lab to a consultant during consulting hours, or (3) show your lab to an instructor at the *beginning* of the next lab session.

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

2. Creating a Blackjack Game

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

Blackjack Rules. 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.

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 `lab09.py` has script code, and so can be safely run as a script. He is a sample transcript showing off a working game:

```
[llee: lab09] python lab09.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
```

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 `lab09.py` take care of all of those details for you.

Fix the method headers. Right now, we cannot do anything with the Blackjack game because we cannot even construct a `Blackjack` object. There is apparently something wrong with some of the method headers. You can tell by running the test script `test09.py` in the command shell. Run the test script now and copy what you see below.

How should you fix the error? Write your fix in the box below.

Now fix *all* method headers that require this correction. We will not ask you to verify that you did this correctly. However, you will not be able to proceed with the lab until you fix it.

Implement and test `__init__`. You should should 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/3/tutorial/datastructures.html

Our solution is three lines long. Write your implementation in the file `lab09.py`. Do not worry about enforcing preconditions just yet, but test your solution with `test09.py` before continuing

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 and create a helper function, as in Assignment 4). Add these assertions to your code in `lab09.py`. If you add a helper function, put it at the top of the file.

3. Scoring the Blackjack Game

Now that you can actually construct a `Blackjack` object, it is time to start implementing the rules. In this case, this just means scoring. And we have already provided you a head start with the method `_score`. Read this method over, but do not change it. Note the leading underscore in this method. This is meant to be a private helper method for the class (and for you).

You should proceed in an iterative fashion to complete the remaining methods in `Blackjack`. For each step outlined in this objective,

(1) Read the directions in this handout and the specification of the relevant methods.
(2) Look at the appropriate test cases in `test09.py` to better understand the goal.
(3) Remove lines with the comment "implement me", and write the appropriate code.
(4) Test your code using `test09.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.

Implement `dealerScore()`. Your implementation should use the private helper method that we have provided. We do not ask you to write it here. Just implement it in `lab09.py`.

Implement `playerScore()`. Your implementation will be very similar to the previous method.

Implement `dealerBust()`. Your implementation should use should use `dealerScore()` as a helper method. Again, we do not ask you to write it here. Just implement it in `lab09.py`.

Implement `playerBust()`. Your implementation should use should use `playerScore()` as a helper method.

Implement and test `__str__`. Note that this method is "higher up" in the file, just after `__init__`, as is conventional. To implement it, you will need to use `dealerScore()` and `playerScore()`. Look at the specification for how to format the string.

Play some Blackjack! This last part is just for fun. Run lab09.py as a script:

```
python lab09.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.

4. 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?