# CS 1110, LAB 9: BLACKJACK

http://www.cs.cornell.edu/courses/cs1110/2014sp/labs/lab09.pdf

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

0.1. **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/cs1110/2014sp/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.

0.2. **New Check-off Policy.** As a solution for the check-off bottleneck in lab hours, we post a new policy starting from this lab 9. You can get a lab check-off at consulting hours or TA office hours by the week's coming Monday (See http://www.cs.cornell.edu/courses/cs1110/2014sp/about/staff.php and note that the instructors will not doing check-offs). This means that we no more do check-off for previous N-1 lab in N lab hours. But surely you can get a new lab checked-off in that week's lab hours. To encourage this, we will release a lab on Sunday evening, so that you can work on it ahead of the lab time. In addition, we will grant you a week of extension for lab 9 thanks to the Spring Break. To summarize this,

- Lab 9 is originally due by Mar 31 (Mon)'s consulting hours or TA office hours, but thanks to the Spring Break we grant everyone an extension by April 7 (Mon)'s.
- We will NOT check lab 9 off in lab 10 hours on April 8-9 (Tu-W).
- We will NOT check lab N-1 off in lab N hours. This is a recurring change that will be applied for the forthcoming labs as well.

But you should always do your best to finish during lab hours. Remember that labs are graded on effort, not correctness.

0.3. **Getting Credit for the Lab.** When done, show your code and/or this handout to a staff member, who will ask you a few questions to see that you understood the material and then swipe your ID card to record your success. Here's a checklist to be ready to finish the lab successfully.

- ☐ You are having a copy of this handout with all of the white boxes filled up at your best.
- ☐ You are ready to show that your modules passes the given test cases by running `python test_blackjack.py` in command shell.
- ☐ You are ready to show that you can play the black jack game by running `python blackjack.py` in command shell.

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

## 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.

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. Here is a sample transcript showing off a working game:

```
[llee: lab09] 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
```

## 2. The *New* Module `card`

We'll be working with a slightly re-defined and re-written version of the Card class from Lab 5.[1] You do not need to change anything in the `card` module. However, since it contains a class

[1] Sustainable computing: reduce and re-use code. (Properly attributing it, of course.) The major changes are that we've removed poker-specific functions and changed the documentation to satisfy the templates we've settled on in lecture.

definition, albeit one whose methods all have "magic names" (unlike in `Blackjack`), you may find it useful to look at as you complete `Blackjack`.

## 3. 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.

3.1. **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.*

3.2. **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:

Why did we originally have poker functions in the module `card`? At the time of Lab 5, it was convenient to put all the required code in one place so that students wouldn't have too many files to deal with, but now we've gotten to the point where exemplifying better file organization is appropriate.

3.3. **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).

3.4. **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:

| |
|---|
| |

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

3.6. **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.

3.7. **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.

3.8. **The checklist.** Again, don't forget the checklist in the first page before you go for a check-off.

3.9. **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?