CS 1110, LAB 05: LISTS; IF-STATEMENTS

http://www.cs.cornell.edu/courses/cs1110/2013sp/labs/lab05/lab05.pdf

L. LEE AND S. MARSCHNER

First Name: ____________________ **Last Name**: ____________________ **NetID**: _____________

Files to download. Create a *new* directory on your hard drive and download the following modules into that directory. (You can get them all bundled in a single zip file at http://www.cs.cornell.edu/courses/cs1110/2013sp/labs/lab05/lab05files.zip)

card.py (http://www.cs.cornell.edu/courses/cs1110/2013sp/labs/lab05/card.py)
cardtest.py (http://www.cs.cornell.edu/courses/cs1110/2013sp/labs/lab05/cardtest.py)
cunittest.py (http://www.cs.cornell.edu/courses/cs1110/2013sp/labs/lab05/cunittest.py)

Getting credit for lab completion. 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.

As always, if you do not finish during the lab, you have until the beginning of lab next week to finish it: show it to your lab TA at the beginning of that next lab. But you should always do your best to finish during lab hours. Remember that labs are graded on effort, not correctness.

---

1. The Module card

You'll be working with the Card type, which is we have almost finished for you in the file `card.py`. Cards have two attributes, a `suit` and a `rank`. It's convenient to encode these as numbers, with two lists serving as "translation tables". To see how this works, open a Python interactive session in the same directory as your lab 05 files, and import the `card` module. Then, for each line below that isn't a comment, enter it (and then type return).

```
card.SUIT_NAMES
 # look at element 0 and 1 of the list card.SUIT_NAMES
card.SUIT_NAMES[0]
card.SUIT_NAMES[1]
```

So, the idea is that int `s` represents the suit given by `card.SUIT_NAMES[s]`, `s` one of 0, 1, 2, 3.

Similarly, try these:

```
card.RANK_NAMES
 # look at some elements of the list card.RANK_NAMES
card.RANK_NAMES[1]
card.RANK_NAMES[2]
card.RANK_NAMES[11]
```

Again, the idea is that int `r` represents the rank given by `card.RANK_NAMES[r]`. [1]

[1] Small detail which you can skip if you don't care: The value `None` in element 0 is a bit of an encoding trick so that we can talk about ranks 1 through 13; there is no card with rank "0".

What happens if you try `card.RANK_NAMES[14]`, and why? Write your answer below.

Now, practice creating and printing some cards. Our constructor for Cards has two[2] parameters, `s` and `r`, the ints for the suit and rank of the new card, respectively.

Try the following:

```
c1= card.Card(0,13)
print card.RANK_NAMES[c1.rank] + ' of ' + card.SUIT_NAMES[c1.suit]
# our method of printing Cards uses essentially the line above
print c1
```

What happens if you try `print card.RANK_NAMES[rank]`, and why? Write your answer below.

Finally, let's practice creating a new list of two new cards. Here's one way to do it: try the following:

```
cardlist = [card.Card(1,4), card.Card(2,11)]
#check that there are exactly two cards
len(cardlist)
print cardlist[0]
print cardlist[1]
```

---

2. Writing Functions for Card Decks and Hands

2.1. print_cards.

print_cards. We're going to be using lists of Cards to represent card decks and card hands. In doing so, we'd like to be able to look at the contents of a list of cards. Try this:

```
print cardlist
```

Not too informative (to humans), right? Your first task is to make a better card-list printing function by completing function `print_cards` in module card.py.

First, open modules card.py and cardtest.py in Komodo Edit. Look at `test_print_cards` in cardtest.py to see that what it does is run `card.print_cards`. Try "Run Python Module" in cardtest.py, and then look at the top of the output; you'll see that nothing gets printed out between the lines "about to print a list of two cards using card.print_cards" and "should have seen...", indicating that `card.print_cards` isn't working yet.

[2]We'll explain why the constructor actually has *three* parameters, if you look at the code, later in the course.

Right now, `card.print_cards` looks like this:

```
def print_cards(clist):
    """Print cards in list clist.

      Precondition:  clist is a list of Cards, possibly empty."""
    for c in clist:
        pass # TODO: fix this line so it prints c (or str(c))
```

The line `for c in clist:` is a very simple version of a *for-loop*, which we'll learn more about in next lecture and subsequently in the course. For now, you can think of this line as saying that it's going to do something for every card `c` that is in the list clist; and what it's going to do to each card is what you say to do to `c` in the next line currently marked TODO.

Change the "TODO" line so that what it does is print the value of variable `c`. This is a two-word statement — should be easy! If you do it right, then run the unit test cardtest, you should see at the top of the output that you got the correct printout.

2.2. **full_deck.** Now you can use `card.print_cards` to see what other functions do. Exit the Python interactive shell, then re-open it and import card.py again.

Here's the specification for function `card.full_deck()`, which has no parameters: "Returns: list of the standard 52 cards". So, try this:

```
fd = card.full_deck()
card.print_cards(fd)
```

Good job with that print_cards function!

2.3. **draw_poker_hand.** OK, now for the exciting (?) part: now that you can play with a full deck, you can try drawing poker hands from it, by implementing `card.draw_poker_hand`. Take a look at its specification. There are going to be several steps involved, which we're going to break up into helper functions.

You'll want to make use of standard list operations to implement your helper functions. Look at section 5.1 here: `http://docs.python.org/2/tutorial/datastructures.html` or the relevant lecture handouts.

2.3.1. *draw.* First, we'll want to be able to randomly draw a single card from a given deck supplied as an argument, and return that card, removing it from the deck.

Look at the code skeleton we've provided you. We've already written a line that choses a random index `i` in the card list given as argument. So all you have to do is add a line or two that both (a) removes the element at index `i`, and (b) returns that element.

Run the unit test cardtest.py and debug as appropriate until your code passes the test of cardtest.test_draw.

2.3.2. *poker_compare.* Now let's deal with the fact that `card.draw_poker_hand` wants an ordering on the list it returns. This ordering will be determined by the function poker_compare.

Using if-statements and the like, implement `card.poker_compare` according to its specification. Note that you'll end up writing some somewhat complex boolean expressions inside your if statements.

Run the unit test cardtest.py to get a quick diagnostic on whether your implementation is correct, by seeing if you get past cardtest.test_compare().

2.3.3. *finish draw_poker_hand using your helper functions.* OK, now for the pièce de résistance: complete function card.draw_poker_hand. Your code should include five calls to draw_card (unless you feel like figuring out that business about for-loops) — each time adding the item returned by draw to the temporary list `output` — and some list functions. There's a hint or two regarding syntax given in the comments in the body of the code.

You can test by running the module *card* (*not* cardtest) repeatedly: you should see a random poker hand printed out each time, plus a small amount of diagnostic information.