CS 1110, LAB 5: LISTS; CONDITIONALS

http://www.cs.cornell.edu/courses/cs1110/2014sp/labs/lab05.pdf

Lab Materials. Create a *new* directory on your hard drive and download the following files into that directory.

`cardstuff.py` (http://www.cs.cornell.edu/courses/cs1110/2014sp/labs/lab05/cardstuff.py)
`cardstufftest.py` (http://www.cs.cornell.edu/courses/cs1110/2014sp/labs/lab05/cardstufftest.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`, and Some Common Error Messages

You'll be working with the Card type, whose class definition is given in file cardstuff.py. Cards have two attributes, a `suit` and a `rank`, both ints. We have these as ints to make it easy to compare card values (for instance, the king of spades may be "worth" more than a two of diamonds). To translate these ints into human-readable card names, we have two lists, defined in module cardstuff, named `SUIT_NAMES` and `RANK_NAMES`; these lists serve as "translation tables". To see how this works, go to the directory you downloaded the lab files into, begin a Python interactive session (type `python` and hit return), and import the cardstuff module at the "`>>>`" Python prompt. Then, for each line below that isn't a comment, enter it (and then type return).

```
print cardstuff.SUIT_NAMES
# the next commands look at element 0 and 1 of the list cardstuff.SUIT_NAMES
print cardstuff.SUIT_NAMES[0]
print cardstuff.SUIT_NAMES[1]
```

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

Similarly, try these:

```
print cardstuff.RANK_NAMES
# the next commands look at some elements of the list cardstuff.RANK_NAMES
print cardstuff.RANK_NAMES[1]
print cardstuff.RANK_NAMES[2]
print cardstuff.RANK_NAMES[11]
```

Thus, int `r` represents the rank given by `cardstuff.RANK_NAMES[r]`. [1]

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

[1]Small detail that you can skip if you don't care: The value `None` in element 0 of `cardstuff.RANK_NAMES` is an encoding trick so that we can talk about ranks 1 through 13, not 0 through 12.

What error message do you get if you enter `print cardstuff.RANK_NAMES[14]` in Python interactive mode, and why? Write your answer below.

Now, practice creating and printing some cards. Our initializer 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 = cardstuff.Card(0,13)
print cardstuff.RANK_NAMES[c1.rank] + ' of ' + cardstuff.SUIT_NAMES[c1.suit]
```

The last line you typed above is what is used inside our definition of the Card class to cause the `print` function to give nice output for cards. To see this, try the following: `print c1`. You should see the exact same output.

What error message do you get if you enter `print cardstuff.RANK_NAMES[rank]` (that is, don't have the `c1` in the command), 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 = [cardstuff.Card(1,4), cardstuff.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.**

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 cardstuff.py.

First, open modules cardstuff.py and cardstufftest.py in Komodo Edit. Look at test_print_cards in cardstufftest.py: you'll see that it calls cardstuff.print_cards on a particular list of cards.

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

Exit Python interactive mode, and run Python on the file cardstufftest.py (`python cardstufftest.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 cardstuff.print_cards" and "should have seen...", indicating that cardstuff.print_cards isn't working yet.

Right now, cardstuff.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 the beginning of a *for-loop*, a construct we briefly mentioned near the end of lecture 8. What it means is that the variable `c` takes on each value in `clist` in turn; for each such value, the body of code indented below the line is executed.

2.2. **full_deck.** Now you can use the cardstuff.print_cards function to see what other functions do. Here's the specification for function cardstuff.full_deck, which has no parameters: "Returns: list of the standard 52 cards". So, try this. Open Python in interactive mode, import cardstuff, and then type these lines.

```
fd = cardstuff.full_deck()
cardstuff.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 (ha), you can try drawing poker hands from it, by implementing cardstuff.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 and comments for `draw` that we've provided you. We've already written a line that choses a random index `i` for the card list given as argument. So all you have to do is add a line or two that (a) removes the element of the argument list at index `i`, and (b) returns that element.

Run the unit test cardstufftest.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 cardstuff.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 `cardstuff.poker_compare` according to its specification. You'll probably need to write some non-trivial boolean expressions inside your if statements.

Run the unit test cardstufftest.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 cardstuff.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 cardstufftest.py repeatedly: you should see a random poker hand, appropriately sorted, printed out near the end each time, plus a small amount of diagnostic information.