CS 1110, LAB 06: LISTS AND OBJECTS: CARDS AND POKER HANDS

http://www.cs.cornell.edu/courses/cs1110/2018sp/labs/lab06/lab06.pdf

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

Getting Credit: **As always, strive to finish during the lab session** — it's the best way to stay on track in this course.[1] You have two weeks due to the prelim, but this lab covers Prelim 1 material, so start it immediately!

1. List Operations

Supposing that the command `lablist=['Cats','rule','!','?','!', '?']` has been executed, complete the following tables as usual, using Python interactive mode to verify whether you were right. Remember: list slicing produces a new list object.

Commands	Expected Output	Were you right?
`lablist.remove('!')` `print(lablist)`		
`lablist.remove('C')`		
`print(lablist[4])`		
`print(lablist[5])`		
`lablist.insert(1, 'may')` `print(lablist)`		
`print(lablist.index('?'))`		
`print(lablist.index('?!'))`		

`copy1 = lablist`
`copy2 = lablist[:]`
`lablist[0] = 'Dogs'`
`print(copy1)`		

`print(copy2)`		
`copy2.append(' And also drool.')` `print(copy2)`		
`print(len(copy2))`		

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

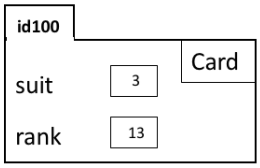
[1] But if you don't manage finish during lab, here are the alternate checkoff opportunities: (a) at ACCEL Green room consulting hours, listed at http://www.cs.cornell.edu/courses/cs1110/2018sp/about/staff.php , **from today until Tue Mar 20 inclusive**, (b) at non-professorial TA office hours **from today to Wed Mar 21 3:45pm inclusive**, although at TA office hours, questions about course material or assignments take precedence over lab check-offs; or (c) during the **first 10 minutes of your next scheduled lab (Tue Mar 20 or Wed Mar 21)**. Beyond that time, the staff have been instructed not to give you credit.
Labs are graded on effort, not correctness. We just want to see that you tried all the exercises, and to clarify any misunderstandings or questions you have.

2. The Lab Files and the Card Class.

Create a new directory and download into it the files you need for lab06 as listed at http://www.cs.cornell.edu/courses/cs1110/2018sp/labs .

We'll work with objects of class Card, defined in file `card_lab06.py`. But we haven't discussed classes in detail yet, so here's all you need to know about that class and how the lab files relate.

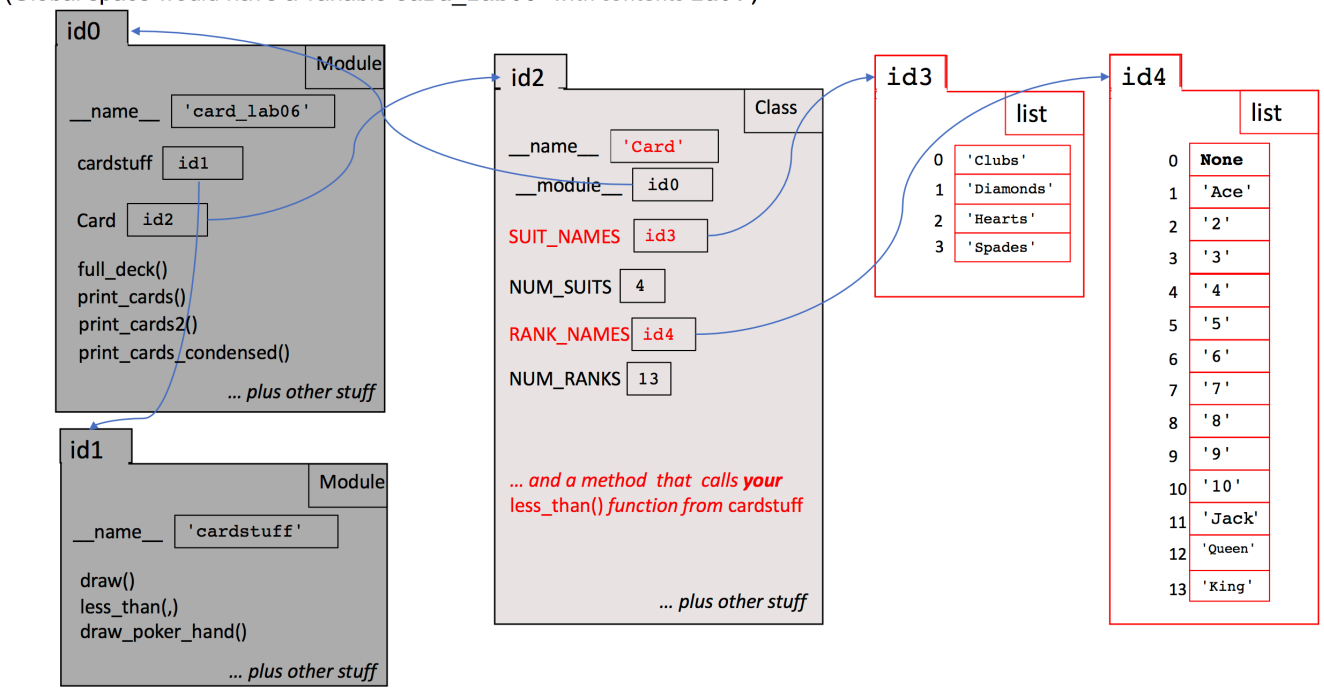
First, Card objects have two attributes, a `suit` and a `rank`, both ints. A constructor expression like `Card(3,13)` creates a new Card object, diagrammed as so (the id number is arbitrary):



Second, the "translation" of a card's `suit` int into an interpretable string is given by the list Card.SUIT_NAMES, which is the list object with identifier `id3` in the diagram below. The "translation" of a card's `rank` int is given by the list Card.RANK_NAMES, which is the list object with identifier `id4` in the diagram below. So, the constructor expression `Card(3,13)` represents a new ♠K card: Card.SUIT_NAMES[3] is `'Spades'` and Card.RANK_NAMES[13] is `'King'`.

In a command shell, navigate to your directory with the lab06 files, begin a Python interactive session, and enter `import card_lab06`. We assert that that `import` statement creates the following situation in heap space (you don't have to understand why):

(You need only focus on the red things in this diagram, which shows the…)
Heap Space after executing `import card_lab06.`
(Global space would have a variable `card_lab06` with contents `id0`.)



3. Working with Card and Lists of Cards

Create a Card and store its id in a variable `c1` with the command `c1 = card_lab06.Card(2,12)`.

What playing card does this correspond to? We'll check by looking at the contents of the two "translation" lists mentioned in Section 2. To reduce the amount of typing needed to access those lists, create aliases for them as follows; enter the following at the Python interactive prompt:

```
suitlist = card_lab06.Card.SUIT_NAMES
ranklist = card_lab06.Card.RANK_NAMES
```

According to the diagram, these commands give `suitlist` the value `id3` and `ranklist` the value `id4`. What do you think you would get by entering `print(ranklist[12] + " of " + suitlist[2])`?

Now, what's the output of the related command `print(ranklist[c1.rank] + " of " + suitlist[c1.suit])`?

The command you entered is essentially what is used[2] inside the definition of class Card to give nice output when a Card is printed or converted to a string via `str()`. To see this, try `print(c1)`. You should see the exact same output.

Next, create a list of two new cards with the command

```
cardlist = [card_lab06.Card(1,4), card_lab06.Card(2,1)]
```

Check that `cardlist` has exactly two cards by: `print(len(cardlist))` — you should get the output `2`. Using the diagram to check the "translations" of suits and ranks, what will be the output of `print(str(cardlist[0]) + "; " + str(cardlist[1]))`?

Check your answer at the Python interactive prompt.

Finally, we look at a function that iterates through a list of cards. Here is a function we've defined for you in `card_lab06.py`:

```
def print_cards(clist):
    """Print cards in clist as a human-readable sequence of lines.

    Precondition: clist is a (possibly empty) list of Cards."""
    for c in clist:
        print(c)
```

The line `for c in clist:` is the header of a *for-loop*. In this loop, variable `c` takes on each value in `clist` in turn; for each such value, the code indented below the line is executed. So, try it out (still in Python interactive mode) to see what it does: `card_lab06.print_cards(cardlist)`.

[2] Employing a little Python "method"-ology we haven't covered in class yet.

Suppose that instead of `print(c)`, the loop body were changed to `print("c")`, like this:

```
def print_cards2(clist):
    """Altered implementation of print_cards just for lab purposes"""
    for c in clist:
        print("c")
```

What output does `card_lab06.print_cards2(cardlist)` give, and why?

Another for-loop is used to to implement function `full_deck` in `card_lab06.py`. Try it out in interactive mode by printing out its result: `card_lab06.print_cards(card_lab06.full_deck())`. And take a look at the actual code when preparing for the prelim.

Now we're playing with a full deck![3]

4. Writing Functions for Card Decks and Hands

Now it's time to try your hand[4] at drawing 5-card poker hands from card decks by implementing function `draw_poker_hand` in file `cardstuff.py`.

Take a look at its specification. There are going to be several steps involved, which we've broken into helper functions. *Make use of standard list operations to implement these helpers* — these can be found in section 5.1 here: http://docs.python.org/3/tutorial/datastructures.html.

First, examine the specification for helper function `draw` in `cardstuff.py`. Notice that we've created a testing function `test_draw` for you in file `cardstufftest.py`. Finish the implementation of `draw` and test it on the command line (`python cardstufftest.py`). Debug if necessary before moving on to the next function.

Next, examine the specification for helper function `less_than` in `cardstuff.py`. Examine the test cases in function `test_less_than` in file `cardstufftest.py` to make sure you understand what the function should do. Write and test this function (via `python cardstufftest.py`).

Finally, the pièce de résistance: complete function `cardstuff.draw_poker_hand`.

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.

[3] (Ha.)
[4] (Ha, ha.)