CS 1110, LAB 11: SUBCLASSES, OR, CRIPPLE MR. ONION

http://www.cs.cornell.edu/courses/cs1110/2017sp/labs/lab11.pdf

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

Updates Monday April 12, 7pm (after the print deadline): see the orange text in §2.

Getting Credit: Deadline: the first 10 minutes of (your) lab **two weeks from now (Tues May 2/Wed May 3)**, due to Prelim 2. But complete this lab as soon as you can — the lab is short, and it covers material that will be on the prelim!

The checking-off procedure is the same as before.[1]

As usual, create a new directory on your hard drive for this lab's files. Then, download into that new directory the files you need for lab 11; get them packaged in a single zip file from the Labs section of the course web page, http://www.cs.cornell.edu/courses/cs1110/2017sp/labs .

1. Reusing the Card class to handle the game "Cripple Mr. Onion"

In several labs, we've used a class Card for representing cards in a standard deck. What about non-standard decks?

The eight-suit card game "Cripple Mr. Onion" appears in some Terry Pratchett novels, and a real-world formulation was created by mathematicians Andrew C. Millard and Terry Tao. The rules are, um, complicated, so we won't implement the game,[2] but we will subclass the Card class to create a new class, OnionCard, which includes the four Latin suits: swords, cups, coins, and staves.

The subclassing here is done as a *convenience* so that we can reuse code already written for the Card class. That is, OnionCards are *not* to be thought of conceptually as instances of regular playing Cards.

To this end, we'll be employing a slightly re-written version of the Card class from a prior lab: we've changed one precondition and one line of code to make re-using the `__init__` and `__str__` "legally"/"morally" OK.[3]

Course authors: E. Andersen, D. Gries, L. Lee, S. Marschner, C. Van Loan, W. White

[1] In case you've forgotten, here's a reminder: Show this handout and/or your code to a staff member either (a) during your lab 11 session, (b) in consulting hours or non-professor office hours listed at http://www.cs.cornell.edu/courses/cs1110/2017sp/about/staff.php up to the day **before** your next scheduled lab section, or (c) in the first 10 minutes of (your) next scheduled lab (Tues May 2/Wed May 3). 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] A stand-alone device on which to play Cripple Mr. Onion was created by Chris Fenton, who describes it as "a device so thoroughly, impractically useless that it's practically just begging to exist".

[3] The previous precondition made reference to `Card.NUM_RANKS`, whereas we want each object to be consulting its own lowest-subclass's `NUM_RANKS`, so we switched to `self.NUM_RANKS`, which will resolve to the class variable `NUM_RANKS` in `type(self)`. A similar situation holds for `__str__`.

2. Part One: Name Resolution

Take a look at the skeleton file onioncard.py. In line 50, you see that we're setting up new suits using class variables in the subclass Card. Given that line, and after checking the relevant parts of the class invariant for class Card in card.py, what will the value of `OnionCard.SUIT_NAMES` be?

Open up a command-line interface in the same directory as you have the lab files. Start up Python, and then at the Python interactive prompt do this:

```
>>> from onioncard import *
>>> from card import * # There was a stray "}" in the printout -- ignore it.
```

What do you get when you try `print Card.SUIT_NAMES`? (It should not be an error.)

What do you get when you try `print OnionCard.SUIT_NAMES`? (It should not be an error.)

Now, quit Python, and then change line 50 of onioncard.py to `SUIT_NAMES = Card.SUIT_NAMES + ['Staves', 'Coins', 'Cups', 'Swords']` . **Restart Python**, and re-import module onioncard. Uh oh; you (should) get an error; why?

Quit Python, and change line 50 back to what it should be.

Now, observe that nowhere in onioncard.py is there an assignment to a variable RANK_NAMES. Given this, predict what will happen when you restart Python and then type:

```
from onioncard import *
print OnionCard.RANK_NAMES
```

Now try it. Why *don't* you get an error; where did the value for RANK_NAMES come from?

3. Part Two: Write the __init__ method for OnionCard

The `__init__()` method for class Card already does what we want the initializer for OnionCard to do. So:

Implement __init__ for OnionCard with a single line that correctly calls the `__init__` method of class Card. What was your one-line implementation?

To test, restart Python, and type in the following:

```
from onioncard import *
regular = OnionCard(1,4)
unusual = OnionCard(6,1)
print str(regular) + ", " + str(unusual)
```

Now, wait a minute: we didn't write a `__str__` method for OnionCard. Why does the print statement above not throw an error?

(Go on to the next page.)

Would you have gotten an error if line 63 in the `__str__` method of **Card** had been

```
return Card.RANK_NAMES[self.rank] + ' of ' + Card.SUIT_NAMES[self.suit]
```

instead of

```
return self.RANK_NAMES[self.rank] + ' of ' + self.SUIT_NAMES[self.suit]
```

?

4. Facilitating checking-off

Here's a checklist to be ready to quickly demonstrate your work to a staff member.

- ☐ You have a copy of this handout with all the white boxes filled in.
- ☐ You have your completed onioncard.py open in Komodo Edit.
- ☐ You have a command shell open in the directory your code is in, so you can demonstrate the running of your code.