CS 1110, LAB 12: SUBCLASSES, OR, CRIPPLE MR. ONION

http://www.cs.cornell.edu/courses/cs1110/2018sp/labs/lab12/lab12.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 2 material, so start it immediately!

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 12; get them packaged in a single zip file from the Labs section of the course web page, http://www.cs.cornell.edu/courses/cs1110/2018sp/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.

2. Name Resolution

Take a look at the skeleton file onioncard.py. In line 44, 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_lab12.py, what will the value of OnionCard.SUIT_NAMES be?

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 May 1 inclusive**, (b) at non-professorial TA office hours **from today to Wed May 2 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 May 1 or Wed May 2)**. 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".

Open up a command shell 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_lab12 import *
```

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 44 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 44 back to what it should be, SUIT_NAMES = cfile.Card.SUIT_NAMES + ['Staves', 'Coins', 'Cups', 'Swords']

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. The __init__ method for OnionCard

The __init__() method for class Card already does what we want the initializer for OnionCard to do. So, we ought to use it. But how should we call it? Below are several possible choices how to replace the "pass" line in __init__, some of which are correct and some of which would cause an error at run-time when you tried to create an OnionCard.

(1) super().__init__(s, r)
(2) super().__init__(self, s, r)
(3) cfile.Card.__init__(s, r)
(4) cfile.Card.__init__(self, s, r)

Which choice(s) are correct? We've given you an easy way to check: replace the "pass" line with a particular choice, save the file, and then, at the command-line (not in Python) run Python on onioncard.py. Doing this executes the command "print(OnionCard(6,10))" and either you get the printout "10 of Cups" or an error occurs.

Hint: according to the Python 3 documentation, super() returns a "proxy" object that delegates method calls to a parent type.

Replace the "pass" statement with the first correct implementation above.

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))
```

You should get the printout 4 of Diamonds, Ace of Cups

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

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.

(That's right, you don't need to show the staff your code.)