Code Clinic Tips

This page include helpful tips based on our experience in helping students in the "Code Clinic" (*interactivepython@online.rice.edu*). Be sure and take a look at these tips if you get stuck.

Tip #1 - Getting started on step one

The program template implements the Card class for you. The class definition includes an initialization method `__init__` that is called whenever we create a card via `Card(..., ...)`, several methods for accessing the fields of a Card object and a method`__str__` that is called when we want to print out information about a Card object.

We have also provided a testing template that calls the provided Card class and includes what the expected output should look like. Cut and paste the Card class code into the template and run the template. If the output from the run matches the output in the comments, the implementation of the Class probably works.

As you implement the Hand class and the Deck classes, you should go through the same testing process for the various methods associated with these classes. Don't rush on to implement the next step without testing first. Note that you don't need to use the canvas to test these methods since we have the string method available to print out the information about class objects to the console.

Tip #2 - Implementing the basic `Hand` class

Step two asks you to implement the methods `__init__`, `__str__`, and `add_card` for the Hand class. Logically, we will think of a hand as a collection cards which we will model in Python as a list of cards. So, to implement the basics of a Hand class, we will need a field in the Hand class to keep track of the list of cards. For the sake of simplicity, let's call that field cards.

- The `__init__` method should create an empty hand by assigning an empty list to the cards field. Implementing this method should be one line of code.
- The `add_card(card)` method should take the Card object `card` and append it to the list in the cards field. Implementing this method should be one line of code.

- The `__str__` method should return a string the includes the string representations for each card (remember you have a string method for Cards to turn a card into a string). Take a look at problem 4 in the practice exercises for mouse and list methods if you are stuck on building this string.

Before proceeding, remember to use the testing template to check whether your Hand class is implemented correctly.

Tip #3 - Implementing the Deck class

Step three asks you to implement methods for the Deck class. Just as in the case of the Hand class, we suggest modeling a deck as a list of cards and keep track of that list using a single field in the Deck class. The method `deal_card` can be implemented in a single line using a common list operation. The `shuffle` method can be implemented using `random.shuffle(...)`. (Remember that`random.shuffle` does not return the shuffled list;a it mutates it's parameter.) You should also implement the `__str__` method to aid in debugging. (This implementation of this method is almost identical to that of the string method for hands.)

The trickiest method is `__init__`which should return a deck containing all 52 cards. To implement `init` for the Deck class, we suggest that you use a pair of nested for-loops or a list comprehension with two for clauses. While building these loops, the first question that you should ask is: "What should these loops be iterating over?". For a deck of cards, the loops should be iterating over the entries in SUITS and RANKS. One possible structure for the loops would be something like:

```
for suit in SUITS:
    for rank in RANKS:
        # create a Card object using Card(suit, rank) and add it to the card list
for the deck
```

Again, test your implementation of the Deck class with the provided testing template before proceeding.

Tip #4 - Hitting a hand using the `Hand` and `Deck` methods

In step four, you are asked to use the `add_card` method for the Hand class and the `deal_card` method for the Deck class to "hit" a Blackjack hand. Given a deck called `my_deck` and a hand called `my_hand`, we can transfer a card from my_deck to my_hand via

```
my_hand.add_card(my_deck.deal_card())
```

Note that this fairly abstract operation is expressed in a single line of code that reflects the logical structure of "hitting" a Blackjack hand. The `Hand` class and the `Deck` class provide us with a layer of abstraction that allows to manipulate hands and deck at a level closer to the true logic of Blackjack.

Tip #5 - Implementing the draw method for a hand

Part of the usefulness of object-oriented programming is that we can use a method from one class to implement a method from another class. In the case of the draw method for a hand, we can use the draw method for card object. The draw method for a hand would look something like

```
def draw(self, canvas, pos)
    for c in self.cards:
        c.draw(canvas,  ...)
        ....
```

The parameter pos will determine where the hand is drawn on the canvas. Note that for this method to work, `c`must be a card object (not a string or tuple). You need to fill in the remaining code (1 line) to position the individual cards in some reasonable pattern based on pos.

Tip #6 - Automated testing

Using the automated testing infrastructure that we have built for the "Fundamentals of Computing" series, we have built some automated unit tests for checking the informal testing templates that are provided in the mini-project description for Blackjack. Both the testing templates and their corresponding automated tests are linked below. To use one of the automated tests, fill in your implementation of the particular class in the testing template, save that program in CodeSkulptor and paste the URL into appropriate field on the unit testing page. **Note that the unit tests expect any auxiliary information like the `Card` class to be included in the submitted code**.

These automated test are more extensive than the manual tests present in the testing templates. You are welcome to post questions in this thread, especially for situations when your code passes the informal tests,

but fails the automated tests. However, please do not post a link to your class implementation in this thread. If necessary, you may email your code to the Code Clinic (interactivepython@online.rice.edu) for assistance.

- `Card` class (just to get started) -- Informal testing template and automated unit test
- Partial `Hand` class -- Informal testing template and automated unit test
- `Deck` class -- Informal testing template and automated unit test
- `Hand` class with `get_value()` -- Informal testing template and automated unit test

Tip #7 - Attribute errors

If you are getting an error like this:`AttributeError: 'str' object has no attribute 'get_rank'`it is (probably) because you've made a Deck and/or a Hand that is full of strings instead of full of Cards. Whenever you create a card you should call the `Card` method, which is the same as saying the `init` method of the Card class. You would say something like

```
    new_card = Card(.....)
```

There are only 2 methods that need this if I recall correctly. One is the `Deck` method that creates a new deck. It must use `Card` to make the cards.The `add_card` method should use the "card" that is PASSED IN. It should NOT create a card in any way.