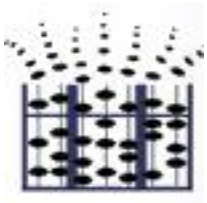

CIS 211
Spring 2014

Project 2: Cards

Due Monday April 14
Upload via Blackboard by 11:00 P.M.

Goals

The first project this term takes a closer look at some of the fundamental concepts of object-oriented programming.

This project has several parts that can be worked on independently, plus several extra credit challenges. Your grade will be based on how many parts you finish and the correctness of each part.

Reading

Introduction to Computing Using Python, sections 6.4–6.5 and 8.1–8.5, plus resources available on the class web site.

Programming Projects

1. Create a file named `Card.py` and enter the definition of a new class named `Card`. Each instance of the class will be a single playing card.

   The constructors should take an integer id between 0 and 51 to specify which card to make. Cards 0 to 12 are clubs, 13 to 25 diamonds, 26 to 38 hearts, and 39 to 51 spades (see the table below for a description of cards and suits).

   Note: this design differs from the one presented in the text. See the class notes for more information.

   Define three accessor functions:
   • `rank()` should return a number between 0 and 12, where 2s have rank 0 and aces have rank 12
   • `suit()` should return the suit number, with clubs = 0, diamonds = 1, hearts = 2, and spades = 3
   • `points()` should return 4 if the card is an ace, 3 if it's a king, 2 if it's a queen, 1 if it's a jack, and 0 otherwise

   Overload the `__repr__` function so the representation of a card is a 2-letter string, and overload the `__lt__` operator so cards are compared according to their ids.

   Here are some examples from an interactive session:

```
>>> from Card import *
>>> x = Card(35)
>>> x
J♥
>>> x.suit()
2
>>> x.rank()
9
>>> x.points()
1
```

2. Define a new class named `BlackjackCard` that uses `Card` as its base class. Overload the `points` method so that aces have 11 points, face cards have 10 points, and other cards have their natural value (9, 8, etc down to 2).

   You should also overload the `__lt__` operator so cards are compared only by their rank, with aces highest, then kings, queens, etc.

```
>>> y = BlackjackCard(38)
>>> y
A♥
>>> y.points()
11
>>> z = BlackjackCard(39)
>>> z
2♠
>>> z.points()
2
>>> y < z
False
```

   Style points: can you figure out how to have `__lt__` use the code you already wrote to compute the rank of a card?

To test the function described in the next project you will need a full deck of cards. The easiest way to make a list of Card objects is to use list comprehension:

```
>>> deck = [Card(i) for i in range(52)]
```

This statement makes a list of Blackjack cards:

```
>>> blackjack_deck = [BlackjackCard(i) for i in range(52)]
```

After you have a deck of cards you can use a function named `sample` from Python's `random` library to deal a hand:

```
>>> from random import sample
>>> hand = sample(deck, 5)
>>> hand
[5♦, 3♦, K♠, J♠, 9♣]
```

3. Define a function named `points` that will take a list of cards and return the sum of the point values of the cards. Here is an example using the hand shown above:

```
>>> points(hand)
4
```

   And here is an example using `BlackjackCard` objects:

```
>>> bj_hand = sample(blackjack_deck,3)
>>> bj_hand
[7♦, 10♦, 9♥]
>>> points(bj_hand)
26
```

   Style points: can you write this function using Python's built-in function named `sum` so it only takes one expression to implement the function?

Extra Credit Ideas

- Have the `Card` constructor check to make sure its argument is between 0 and 51 and raise an exception otherwise.
- Allow users to pass a specified rank and suit to the `Card` constructor, for example have `Card('A','\u2660')` return the same object as `Card(51)`. Another variation is to allow users to pass a single string, e.g. `Card('A♠')`.
- Write a function named new_deck that will create a list of cards of a specified type. The argument should be the name of the class that specifies which type of card to make (if no argument is passed make a list of standard cards):

```
>>> new_deck(Card)
[2♣, 3♣, 4♣, … Q♠, K♠, A♠]
>>> new_deck(BlackjackCard)
[2♣, 3♣, 4♣, … Q♠, K♠, A♠]
```

Testing

See the lecture notes for an explanation of how Python's unittest module will check your code.

Note: `test_Cards.py` must be in the same directory as your `Card.py` file.

We will test your classes and functions by running the tests in a file named `test_Cards.py`. You can download the file and run the tests yourself before you submit your project. This is what you will see in your shell window when your code passes all the tests:

```
$ python3 -m unittest test_Cards.py
...
----------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

Card Names and Symbols

This table shows the name and suit symbols used in a standard deck of playing cards. For suits you can use Unicode symbols or (if your terminal window or IDE is having issues with Unicode) just print a single letter.

Card names use the numbers 2 through 10, and then letters J (for jack), Q (for queen), K (for king), and A (for ace). Here are the first 13 cards in the deck, showing each card name:

```
>>> [Card(i) for i in range(13)]
[2♣, 3♣, 4♣, 5♣, 6♣, 7♣, 8♣, 9♣, 10♣, J♣, Q♣, K♣, A♣]
```

Suits are clubs (♣), diamonds (♦), hearts (♥), and spades (♠). This expression shows the ace of each suit:

```
>>> [Card(i) for i in range(12, 52, 13)]
[A♣, A♦, A♥, A♠]
```

What to Turn In

Documentation Write a short description (two or three paragraphs total) of what you did for this project. This is your chance to tell the graders about anything you think you did well. The documentation should be in a file named `writeup` with an extension that identifies the file format (.doc for Microsoft Word, .pdf for Adobe PDF, .txt for plain text, .rtf for rich text format).

Code Save all your class and function definitions in a single file named `Card.py`.

Create a package (tar or zip format) that includes your writeup and code and upload the package via Blackboard.