UC-Berkeley-I-School / **mids-w200-assignments-upstream-fall2021** Private

Code Issues Pull requests Actions Projects Wiki Security

main

mids-w200-assignments-upstream-fall2021 / week_07 / **HW_Unit_07.ipynb**

fosterrj Added week07 activity and HW **History**

1 contributor

```
1258 lines (1258 sloc) | 37.3 KB
```

Week 7 Assignment - W200 Introduction to Data Science Programming, UC Berkeley MIDS

Write code in this Jupyter Notebook to solve each of the following problems. Each problem should have its solution in a separate cell. Please upload this **Notebook** with your solutions to your GitHub repository in your SUBMISSIONS/week_07 folder by 11:59PM PST the night before the Unit 9 class (you have two weeks to complete this assignment).

Objectives:

- Demonstrate how to define classes
- Design and implement class objects and class interactions
- Understand how to call methods from both inside and outside of classes
- Understand how to set internal attribute within a class

General Guidelines:

- All calculations need to be done in the classes (that includes any formatting of the output)
- Name your classes exactly as written in the problem statement
- Do NOT make separate input() statements. The classes will be passed the input as shown in the examples
- The examples are using the '>>>' as the command entered into a Jupyter coding cell with the example output shown below it. Your program will be run on other tests not just the example provided.
- The examples given are samples of how we will test/grade your code. Please ensure your classes output the same information
- Answer format is graded - please match the examples
- Inputs to classes and methods do need to be validated or checked where the problem or examples specifically state to check for inputs. Otherwise, you can assume that the correct input will be sent as shown in the examples.
- Comments in your code are strongly suggested but won't be graded
- This homework is mostly auto-graded. The blank code blocks are the auto-grading scripts - please do not delete these!
- Your code needs to be written in the #Your Code Here blocks or it wont be graded correctly.
- `__str__` and `__repr__` methods are discussed in Unit 8.

Project Proposal

Reminder! Please complete your project proposal, as discussed in class and outlined in the project_1 folder. Submit your 1-2 page proposal as a PDF in your SUBMISSIONS repo under the project_1 folder.

This is worth 10 points of your **project** grade (not the grade for this homework).

7.1 A Quick Reading (20 points)

7-1 A Quick Reading (20 points)

Please read the following article and write a couple sentences (~100 words) of reaction. What is the most interesting part?

Write code that is easy to delete, not easy to extend (http://programmingisterrible.com/post/139222674273/write-code-that-is-easy-to-delete-not-easy-to)

This article is to explain an "architectural" perspective towards thinking about writing in large code bases. This might not really apply to the work that you are doing now but should provide some food for thought on upcoming projects. Think about the author's perspective and why he seems to have come to it. Please don't worry about knowing all the terminology or programs/systems that he refers to. We want you to extract what he's trying to say about writing code rather than the intricacies of the low-level systems that is referring to.

YOUR ANSWER HERE

Please do two out of the three parts below (80 points - 40 points each)

That is, please do two of the three questions (7-2, 7-3 & 7-4)!

If you want to do all three parts please write a comment on which two parts to grade. If a comment isn't found, Questions 7-2 & 7-3 will be graded.

7-2 Deck of Cards

Please design two classes in this notebook as follows:

1. Please create a class called **PlayingCard**. This class should have:

- An attribute, "rank" that takes a value of "2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", or "A"
- An attribute, "suit" that takes a value of "♠" "♥" "♦" or "♣". (If you don't know how to make these characters you can cut and paste from this block)
- An **init** function that:
    - Accepts as parameters a specific rank (as a string) and suit (as a string).
    - Gives an appropriate response when a rank or suit is not valid.

2. Please create a class called **Deck**. This class should have:

- An attribute, "cards", that holds a list of PlayingCard objects.
- An **init** function that:
    - By default stores a full deck of 52 playing card (with proper numbers and suits) in the "cards" list. Each cards will be of the class PlayingCard above

    - Allows the user to specify a specific suit (of the 4 - "♠" "♥" "♦" or "♣"). In this case, the program should only populate the deck with the 13 cards of that suit.
    - After the cards object is initialized, call the "shuffle_deck()" function (below).
- A "shuffle_deck()" function that randomly changes the order of cards in the deck.
    - You can import the random library to 'shuffle' the deck: https://docs.python.org/3.7/library/random.html (https://docs.python.org/3.7/library/random.html)
    - If you import random, please import it at the top of your block instead of inside the class / methods.
- A "deal_card(card_count)" function that removes the first card_count cards from the deck and returns them as a list.
    - Make sure this function gives an appropriate response when the deck is out of cards.

3. You might have to write `__str__` or `__repr__` methods to display the cards correctly in either or both classes.

Example:

```
>>> card1 = PlayingCard("A", "♠")
>>> print(card1)
A of ♠

>>> card2 = PlayingCard("15", "♠")
Invalid rank!

>>> card2 = PlayingCard("10", "bunnies")
Invalid suit!

>>> deck1 = Deck()
>>> print(deck1.cards)
[K of ♠, A of ♥, 6 of ♣, 7 of ♠, J of ♦, 6 of ♠, Q of ♦, 5 of ♣, 10 of ♦, 2
 of ♥, 8 of ♣, 8 of ♦, 4 of ♦, 7 of ♦, 3 of ♣, K of ♣, 9 of ♠, 4 of ♥, 10 of
♥, 10 of ♣, A of ♠, 9 of ♥, 7 of ♥, 9 of ♣, 7 of ♣, 5 of ♠, 3 of ♦, 10 of ♠,
Q of ♥, J of ♣, 5 of ♥, K of ♥, K of ♦, 2 of ♠, 8 of ♠, Q of ♣, 3 of ♠, 6 of
♥, 6 of ♦, A of ♣, A of ♦, 3 of ♥, J of ♠, 4 of ♣, 5 of ♦, 2 of ♦, 4 of ♠, 2
of ♣, Q of ♠, J of ♥, 8 of ♥, 9 of ♦]

>>> deck2 = Deck('♠')
>>> deck2.shuffle_deck()
>>> print(deck2.cards)
[A of ♠, 10 of ♠, 3 of ♠, 7 of ♠, 5 of ♠, 4 of ♠, 8 of ♠, J of ♠, 9 of ♠, Q
 of ♠, 6 of ♠, 2 of ♠, K of ♠]

>>> deck2.deal_card(7)
[A of ♠, 10 of ♠, 3 of ♠, 7 of ♠, 5 of ♠, 4 of ♠, 8 of ♠]

>>> deck2.deal_card(7)
Cannot deal 7 cards. The deck only has 6 cards left!
```

In [ ]:
```
# YOUR CODE HERE
```

In [ ]:
```
# Autograde cell - do not erase/delete
# Here we implement some tests for you to compare your code with the ou
tput we expect.
import random
import subprocess
from nose.tools import assert_equal
from nose.tools import assert_true

# String representation of PlayingCard class matches expected output?
print(PlayingCard('A', '♠'))
print(str(PlayingCard('A', '♠')) == 'A of ♠')
```

In [ ]:
```
# Autograde cell - do not erase/delete
# Here we implement some tests for you to compare your code with the ou
tput we expect.

# Does your deck class initialize as expected?

D = Deck("♠")

solution = ['A of ♠', '8 of ♠', '4 of ♠', 'K of ♠', 'J of ♠',
            '5 of ♠', '6 of ♠', '9 of ♠', 'Q of ♠', '3 of ♠',
            '7 of ♠', '10 of ♠', '2 of ♠']

for i in D.cards:
    test = str(i) in solution
    print(str(i) + ' in solution list?', test)

    if test:
        solution.remove(str(i))

print(10 * '---')
print('Are any cards left in the deck?', solution)
```

In [ ]:
```
# Autograde cell - do not erase/delete
# Here and in the following cells we implemented some hidden tests.
# This means: Check your code for other inputs as well! Does it match w
ith what you would expect?
# Test the other methods you implemented. Do they work as expected?
```

In [ ]:
```
# Autograde cell - do not erase/delete
```

In [ ]:
```
# Autograde cell - do not erase/delete
```

In [ ]:
```
# Autograde cell - do not erase/delete
```

In [ ]:
```
# Autograde cell - do not erase/delete
```

In [ ]:
```
# Autograde cell - do not erase/delete
```

In [ ]:
```
# Autograde cell - do not erase/delete
```

In [ ]:
```
# Autograde cell - do not erase/delete
```

Extra Credit (2 points): Write a method called **war** in the coding cell above under the Deck class that deals a card to the player and a card to the dealer from your deck. Whomever has the highest ranked card wins; print them a nice message! (2 is the lowest rank and A is the highest). If it is a tie, print a different message. In the cell below, show three examples of running your **war** method.

In [ ]:
```
# YOUR CODE HERE
```

7-3 Galton's Box

The following figure depicts Galton's box, a game in which marbles are dropped through N rows of pins. In row 0, there is one position a marble can be in (labeled 0), in row 1, there are two positions (labeled 0 and 1), and so forth. Each time the marble bounces from one row to the next, there is a 50% probability it bounces left and a 50% probability it bounces right.

Notice that if a marble is in position x of row y, and it bounces left, it ends up in position x of row y+1. If it bounces right, it ends up in position x+1.

1. Create a class, **Marble**, to represent a single Marble that will drop through Galton's Box.

- Include attributes to represent the position of the marble.
- The `__init__` method should accept a one-character label for use when printing the Marble.

2. Create a class, **GaltonBox**, to represent the overall setup. You should include the following methods:

- `__init__` - Your initializer should accept the size of the box (number of rows including the start row), N.
- `insert_marble` - This method should accept a Marble instance and sets its position to position 0, row 0.
- `time_step` - This method should cause all Marbles in Galton's box to bounce to the next row, dropping left or right with equal probability. When a marble reaches row N-1 at the bottom of the box, it should not move any more. Note that you should simply allow marbles to occupy the same position (instead of working out a system to prevent a Marble from entering a position if another Marble is already there).
  - You can import the random library to decide which way the marble bounces: https://docs.python.org/3.7/library/random.htm (https://docs.python.org/3.7/library/random.htm)
  - If you import random, please import it at the top of your block instead of inside the class / methods.
- `__str__` and `__repr__` - Include methods to display the Marbles currently in the box. To keep things simple, if there are multiple Marbles in a given position, you only have to display one of the labels.

Your classes should mimic the following behavior (except that the horizontal positions are random):

```
>>> m1 = Marble("x")
>>> m2 = Marble("o")
>>> box = GaltonBox(3)
>>> box.insert_marble(m1)
>>> box
x
--
---
>>> box.time_step()
>>> box
-
-x
---
>>> box.insert_marble(m2)
>>> box
o
-x
---
>>> box.time_step()
>>> box
-
o-
-x-
>>> box.time_step()
>>> box
-
--
ox-
```

In [ ]:
```
# YOUR CODE HERE
```

In [ ]:
```
# Autograde cell - do not erase/delete
# Here we implement some tests for you to compare your code with the ou
tput we expect.

# Does the string representation match?

box = GaltonBox(3)
box.insert_marble(Marble("o"))
print(box)
print(str(box) == 'o\n--\n---\n')
```

In [ ]:
```
# Autograde cell - do not erase/delete
# Here we implement some tests for you to compare your code with the ou
tput we expect.

# Does the time-step method work as expected?
import random
random.seed(1234)
```

```
box.time_step()
print(box)
print(str(box) == '-\n-o\n---\n')
```

In [ ]:
```
# Autograde cell - do not erase/delete
# Here and in the following cells we implemented some hidden tests.
# This means: Check your code for other inputs as well! Does it match w
ith what you would expect?
# Test the other methods you implemented. Do they work as expected?
```

In [ ]:
```
# Autograde cell - do not erase/delete
```

In [ ]:
```
# Autograde cell - do not erase/delete
```

In [ ]:
```
# Autograde cell - do not erase/delete
```

In [ ]:
```
# Autograde cell - do not erase/delete
```

In [ ]:
```
# Autograde cell - do not erase/delete
```

Bonus Investigation: (Extra Credit: 2 points) Once your code is working, write code to create a box with 20 rows, insert a 100 Marbles, and repeatedly call time_step() until all Marbles are at the bottom. Now adapt the following code to display a histogram of the final Marble positions. What does the shape of the distribution look like?

In [ ]:
```
# Sample code below - you will have to adjust it to work for your code.

%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np

x_positions = (2,3,4,6,7,4,3,2,3,1)
cutoffs = np.arange(min(x_positions) - .5, max(x_positions)+.5)
plt.hist(x_positions, bins = cutoffs)
```

In [ ]:
```
# YOUR CODE HERE
```

YOUR ANSWER HERE

7-4 Sorting Marbles

In a particular board game, there is exactly one row and it comprises N spaces, numbered 0 through N - 1 from left to right. There are also N marbles, numbered 0 through N - 1, initially placed in some arbitrary order. After that, there are two moves available that only can be done one at a time:

- Switch: Switch the marbles in positions 0 and 1.
- Rotate: Move the marble in position 0 to position N - 1, and move all other marbles one space to the left (one index lower).

The objective is to arrange the marbles in order, with each marble i in position i.

1. Write a class, **MarblesBoard**, to represent the game above. The class should be initialized with a particular sequence of Marbles.

- Write an `__init__` function that takes a starting sequence of marbles (the number of each marble listed in the positions from 0 to N - 1). (Notice in the sequence all the marbles are different numbers and are sequential numbered but not in order!)
- Next, write switch() and rotate() methods to simulate the player's moves as described above.
- Write a method, is_solved(), that returns True if the marbles are in the correct order, False otherwise.
- Additionally, write `__str__ and __repr__` methods to display the current state of the board.

Your class should behave like the following example:

```
>>> board = MarblesBoard((3,6,7,4,1,0,8,2,5))
>>> board
3 6 7 4 1 0 8 2 5
>>> board.switch()
>>> board
6 3 7 4 1 0 8 2 5
>>> board.rotate()
>>> board
3 7 4 1 0 8 2 5 6
>>> board.switch()
>>> board
7 3 4 1 0 8 2 5 6
```

2. Write a second class, **Solver**, that actually plays the MarblesGame.

- Your class will take a MarblesBoard in its initializer.
- Write a solve() method that repeatedly calls the switch() and rotate() methods of the given