UNIVERSITY OF WATERLOO

A new course announcement has been posted. Click **here** to view the new announcement.

Bridge - Dealing Cards

Additional Instructions

Deadline

This project was posted on July 28th and is due on August 11th at 10:00am. Late submissions will not be accepted.

Discussion Forums and Office Hours

- All discussion forum posts will be private as of July 28th.
- Any clarifications that are provided will be added in announcements. This is to ensure fairness for all students and that all students have the same information. Please refresh the page periodically to make sure you are getting new updates.
- Office Hours will be held as normal until July 30th. However questions pertaining to the final project can only be answered by instructors and will not be answered by non-instructor course staff.
- August 2nd is a holiday and no office hours will be held that day. Following this, instructors will have the following set of office hours (all in ET):
    - Tuesday August 3rd: 8-10am [Stacey]
    - Wednesday August 4th: 1-3pm [Stacey]
    - Thursday August 5th: 8-10am [Stacey]
    - Friday August 6th: 10am-12pm, 1-2pm [Carmen]
    - Monday August 9th: 8-10am [Stacey] and 10:30-11:30am [Carmen]

- Tuesday August 10th: 8-10am [Stacey]

Note that accommodations can be attempted to be made if the above do not work. Please email your instructor on record to see if another time can be arranged.

- Instructors will answer questions about how Bridge works or about any concepts from the modules. However, note that we will **not** be reading your code to determine if you have bugs or errors. It is your responsibility to make sure your code works.

Grading

- All marks will be based on correctness, that is, that your code passes our tests. However marks can still be deducted if code does not meet the instructions on the general instrruction page.
- While no marks are allocated for design recipe steps, you are encouraged to add purpose and contract statements for all helper functions you write.
- You will need to design your own tests to be confident that your project is correct, but these also will not be marked. We provide some tests below but passing these test does not ensure your code is correct.
- If you pass all the basic [public] tests on MarkUs, you will receive at least 50% on the project. However you will not know what these tests are (only that you have passed/failed them).
- Note that even if you are unable to get all the public tests, you can still receive a passing grade based on your work in the parts you have successfully completed. That is, partial marks will be awarded based on which tests you pass.
- It will be critical for you to submit often, and to check your basic test results. There is no penalty for submitting early and often and we encourage you to do so.
- Note that we will **not** be providing any details or answering any questions about these basic tests, other than the name of the function being tested. Part of the requirements of this assessment is understanding what situations should be tested.
- A solution set will not be posted for this project. Your results on the project will be available through MarkUs, as with the term assignments.
- There are lots of opportunities for partial marks so even if you cannot get every function below working, there is still an opportunity to do well in this project. The most important part is to start early, test frequently and submit often.

- We mark your **last submission** before the deadline so be careful of any last minute changes without running your own tests and checking the basic tests.
- Please note that neither public nor private tests will be given out. In fact, public tests will only give you the information of being True (ie. passing) or False (ie. failing). Do not read into the boolean values other than that you have passed or failed a public test.

Warning - Read before you start coding!

- The project is about the card game bridge. We encourage you to familiarize yourself with the game by watching a video first (or reading the entire final project) and then playing some hands against computer opponents. Our recommended sites to explore and play include:
    - A Youtube Video explaining the rules of the game https://youtu.be/2IomnCvxWzM
    - Bridge Base Online https://www.bridgebase.com/
    - Funbridge https://www.funbridge.com/
- There is a lot of material to read here. We strongly advise you to read the project in its entirety including reading through the provided code. Do this at least twice.
- **Don't forget to always check your email for the public test results after making a submission!**
- Lastly, try to take your time. This project will likely require several readthroughs and going back and forth between required functions and parts. Remember that anything you code in earlier sections can and should be used in later ones! In later modules, we allow you to use functions from pervious parts on EdX by embedding our (hidden) implementation of the functions into the code. Meaning, as an example, in the third part of this project, we import the solutions from the first and second question so you can write code that uses these functions as though the solution was given to you.

One Last Reminder

> WARNING: For the purposes of Academic Integrity, the final project is treated like a final exam. This means penalties are increased for any academic offences (including, but not limited to

receiving 0 in the entire course!) You have been warned.
Reminder: Do not discuss this assignment with anyone except the instructors on the EdX discussion forums!

Question 1

In this Final Project, you will write a program to play the game Bridge.

Bridge is a trick-taking card game played with 4 people in two teams of two and using a standard deck of 52 cards (13 cards: 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A in four suits: clubs, diamonds, hearts and spades; the J, Q, K, A represent the Jack, Queen, King and Ace). If you wanted to fool around with a website to play against some computer players, try Bridge Base Online. The basic rules of the game will be explained here enough for you to get started learning the game. Like many other great games, it takes only a bit of time to learn but a lifetime to master.

Your final project is broken up into several components: dealing cards, the bidding, the playing and the scoring. By the end you will have a fully working version of the game that could work on one computer.

Preliminary Definitions

- A **hand** consists of the cards a player holds.
- A **trick** is the set of 4 cards that are played in the middle of the table, one from each player, and taken by the player who plays the highest card (more on this in the next part). Cards will be dealt evenly meaning the play of bridge will involve 13 total tricks.

Dealing

As mentioned above, bridge is a card game played with a standard deck of 52 cards (13 cards: 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A in four suits: clubs, diamonds, hearts and spades). Part of your goal will be to complete the `Card` class below along with any class methods/functions that are incomplete and the `Player` class we will use in the next sections:

```
218
219 def deal_bootstrap(deck = []):
220     '''
221     Simulate a deal of a bridge game woth North, East, South and West.
222     Optional parameter deck shoudl be a permutation of numbers from 1 to 52
223     to be a proper simulation. Can have smaller size and repeats however if
```

```
    desired.

    deal_bootstrap: [(listof Nat)] -> (list Player Player Player Player)
    Requires:
        1 <= deck[i] <= 52 for all indices i
    '''
    invalid_response = "Invalid response."
    random_prompt = "Do you want to use a (r)andom deal or a (p)redefined deal
    random_seed = \
       "Do you want to use a fixed seed? Enter (n)o or a natural number: "
    dealer_prompt = "Who is the dealer? (N)orth, (E)ast, (S)outh, (W)est? "
    dealer_dict = {'N': 'North', 'S': 'South', 'E':'East', 'W': 'West'}
    PLAYERS = list(dealer_dict.values())

    random = input(random_prompt)
    while random not in ['r', 'p']:
      print(invalid_response)
      random = input(random_prompt)
    if random == 'r':
      s = input(random_seed)
      while s != 'n' and not s.isnumeric() and int(s) < 0:
        print(invalid_response)
        s = input(random_seed)
      if s == 'n':
        s = None
      else:
        s = int(s)
      deck = shuffle(s)
    else:
      deck = list(map(convert_to_card, deck))
    dealer = input(dealer_prompt)
    while dealer not in ['N', 'E', 'S', 'W']:
      print(invalid_response)
      dealer = input(dealer_prompt)
    dealer = dealer_dict[dealer]

    ##Dealer is last player in list to work with dealer:

    player_index = PLAYERS.index(dealer)
    player_names = PLAYERS[player_index + 1:] + PLAYERS[:player_index+1]
    player_list = list(map(lambda x: Player(x, []), player_names))

    deal(deck, player_list)
    return player_list
```

a. Submit and complete the `Card` magic method

`__init__(self, val, st)`

to set the class variables `value` and `suit` for a card.

b. Code the `Card` magic method

```
__eq__(self, other)
```

that determines when `Card`s are equal. Two `Card`s are considered equal if they have the same `value` and `suit`.

c. Write the code for the `Player` magic method

```
__eq__(self, other)
```

that determines when `Player`s are equal. Two `Player`s are considered equal if they have the same name and their hands are equal up to a reordering of the `Card`s in their `hand`s.

d. Write the code for the `Player` class method

```
play_card(self, card)
```

that will remove a card from a player's hand if it exists and return `True`. Otherwise, the function does not mutate anything and returns `False`.

e. Submit and complete the function

```
convert_to_card(n)
```

that will consume a natural number between 1 and 52 inclusive and return the corresponding `Card` according to the scheme below

- The numbers from 1 to 13 will represent clubs.
- The numbers from 14 to 26 will represent diamonds.
- The numbers from 27 to 39 will represent hearts.
- The numbers from 40 to 52 will represent spades.

As for the values of cards, the numbers 1, 14, 27, 40 will represent the value `'A'`, the numbers 2, 15, 28, 41 will represent the value `'2'` and so on in order up to the numbers 13, 26, 39, 52 will represent the value `'K'`. It is perhaps a bit confusing to give the value of the Ace the smallest value despite as we will see it is the highest card in a suit, however, the Ace in many games is also the lowest valued card (but not in Bridge) and many programmers keep with this convention.

f. Code the function

```
deal(cards, players)
```

that will distrubute the `cards` as evenly as possible amongst `players`, a list of `Player`s. The function should mutate the `player[k].hand` field for each `k` where the first card goes to the first player (in position 0 in `players`), the second card goes to the second player and so on.

g. Write the code for the function

```
display_hand(hand)
```

that will print out the cards in a nice format. The cards should be in descending order with aces in the left most position followed by any kings all the way down to twos. If a suit is not present, the suit symbol should be printed followed by a space followed by a dash '-' (no quotes). Suits can be printed by calling `chr(suit)` where `suit` is one of `9827, 9830, 9829, 9824` for clubs, diamonds, hearts and spades respectively. Each suit should be on its own row and each row ends with a single newline character (the default one that is given by `print` is fine).

A function `shuffle` is provided to you in case you want to test some random values. This uses the `numpy` package, an extremely powerful computing module that we will unfortunately not have the time to explore in this course. It consumes a default parameter `seed` which should be set to an integer if you want a random, but consistently random value (otherwise, leave this parameter blank or pass `None` to get a truly random set of values). This will deal all the cards which will be useful in future parts if you want to play some sample hands. This function will not work until you complete the `convert_to_card` function above. There is also `deal_bootstrap` which can consume a optional parameter which should be a random list of distinct numbers between 1 and 52 that can be used to see if your deal works correctly.

Last submission on 04-08-2021 00:49:57 Result: Could not find test results

main.py - Add File

```
311
312 ##Tests play_card:
313 p = Player("North", [Card("A", "C"),
314                       Card("2", "C"), Card("3", "C")])
315 check.expect("Test play_card True", p.play_card(Card("A", "C")), True)
316 check.expect("Test play_card True Mutation", p.Player("North", [Card("2", "
```

```
check.expect("Test play_card True Mutation", p, Player("North", [Card("2",
                                                    Card("3", "C")]))

p = Player("North", [Card("A", "C"),
                     Card("2", "C"), Card("3", "C")])
check.expect("Test play_card False", p.play_card(Card("7", "C")), False)
check.expect("Test play_card False No Mutation", p, Player("North", [Card("A
                                                    Card("2", "C"),
                                                    Card("3", "C")]))

##Tests deal:


P = [Player("North", []), Player("East", []), Player("South", [])]
check.expect("Test deal Simple", deal([], P), None)
check.expect("Test deal Simple no mutation", P,
             [Player("North", []), Player("East", []), Player("South", [])])

L = [Card("A", "C"), Card("2", "C"),
     Card("3", "C"), Card("4", "C")]
P = [Player("North", []), Player("East", [])]

check.expect("Test deal Simple", deal(L, P), None)
check.expect("Test deal Simple Mutation", P,
             [Player("North", [Card("A", "C"), Card("3", "C")]),
              Player("East", [Card("2", "C"), Card("4", "C")])] )

L = [Card("A", "C"), Card("2", "C"),
     Card("3", "C"), Card("4", "C") , Card("5", "C")]
P = [Player("North", []), Player("East", []),
     Player("South", []), Player("West", [])]

check.expect("Test deal Simple", deal(L, P), None)
check.expect("Test deal Simple Mutation", P,
              [Player("North", [Card("A", "C"), Card("5", "C")]),
               Player("East", [Card("2", "C")]),
               Player("South", [Card("3", "C")]),
               Player("West", [Card("4", "C")])])


##Test for display_hand

L = [Card("A", "C"), Card("10", "C"), Card("4", "S")]
check.set_print_exact("♠ 4", "♥ -", "♦ -","♣ A 10")
check.expect("Test example", display_hand(L), None)
```

Code Output

Run Code | Save Code | Reset Code | Show History

Visualize

Submit Code | View Submissions

Discussion

Hide Discussion

Topic: Module_10 / Final Project: Bridge - Dealing Cards

Add a Post

Show all posts by recent activity

There are no posts in this topic yet.