CMPSC 132 Final Project: Blackjack

Notes:

- As a reminder, this project is worth 25% of your final grade. Project is due on 05/01/2020 at 11:59 PM EST. Please note that the sooner you submit your project, the faster we can compute your final grade
- Download the starter code files from the Final Project Assignment on Canvas. Do not change the **file names**, function names or given started code on your script.
- A doctest is provided as an example of code functionality. Getting the same result as the doctest does not guarantee full credit. You are responsible for debugging and testing your code with other cases
- To submit your 4 files on Gradescope, you have to drag-drop all 4 files at the same time (or select all 4 if you are using the Upload Box). Failing to do this, will result in 3 resubmissions of your assignment (one file per resubmission) instead of one submission containing all your files.

Blackjack Overview (Modified Rules)

For the final project in this class, you will be implementing a simulation of the game a blackjack, with some variations.

Blackjack is a card game which is played by the dealer and one or more regular players using a standard deck of cards. The rules that you will be implementing are a variation of the original rules, so please read the following carefully! The modified rules of the game are as follows:

Objective

The goal of the game is to accumulate cards such that the sum of cards in a player's hand is as close to 21 as possible without going over 21. A player wins by beating the dealer (i.e. having a higher total than the dealer which is still less than 21). If the total of any player or the dealer goes over 21, that person has "gone bust" and automatically loses for that round.

Card Values

Each card is worth the amount of the card's value, independent of suit (e.g. the 5 of spades adds 5 to your total, the 3 of hearts adds 3 to your total). Face cards (Jack, Queen, King) are worth 10, and Aces are worth 11.

Dealing:

At the beginning of each round of play, each player first receives a card, then the dealer receives his card. Then each player receives a second card, and the dealer receives a second card.

Natural Blackjack

If any player's first two cards total 21 (only possible if a player receives an ace and a card with a value of ten as his first two cards), this is called a "natural blackjack." If the dealer does not have a natural blackjack, then any player who does have a natural blackjack is awarded a win, and play continues as normal for the other players. If the dealer has a natural blackjack, then any players who also have a natural blackjack are awarded a tie, and any players who don't have a natural blackjack are awarded a loss, and the round is over.

Playing The Hand

Each player, in order, will have the opportunity to improve his total. A player can either "stand," which means he doesn't want any more cards and is ending his turn, or he can "hit," which means he is asking for an additional card from the deck. A player can hit as many times as he wants as long as his total stays under 21. As soon as a player's total equals exactly 21, he should stop hitting since 21 is the highest achievable total without going bust. If a player's total becomes larger than 21, he has gone bust and can no longer hit.

After all of the players have ended their turns (either by standing or by going bust), the dealer then plays. The dealer plays in a pre-determined manner. If the dealer's total is 16 or less, he must hit. If the dealer's total is 17 or greater, he must stand.

Settling The Hand

After the dealer's turn has ended, the hand is over and winners are determined. If the dealer went bust, then any player who did not also go bust is awarded a win (players who went bust are awarded a loss). If the dealer did not go bust, any player with a total higher than the dealer is awarded a win, any player with a total equal to the dealer's total is awarded a tie, and any player with a total lower than the dealer (or that went bust) is awarded a loss.

Implementation

Please read these class descriptions carefully.

Testing note: to run the doctests provided in the starter code, you should execute the command `python3 <filename>.py -v` (Mac/Linux) or `python <filename.py> -v` (Windows) instead of the usual command `python3 -m doctest -v <filename>py.` Thus if you want to test the `player.py` file you should run `python3 player.py -v`

The methods that you are expected to implement all contain `return None` as the method body by default. This allows you to run doctests before you've completed all the methods. Feel free to remove/replace this `return None` when you actually implement each method, as it only serves as a placeholder.

It's important to note that the doctests provided in the starter code do not necessarily always reflect how a method will be used in the context of the game. These doctests are instead meant to describe the correct behavior of methods in isolation, without considering the rest of the project. Just because a method is used in a certain way in a doctest does not mean that the method should be used in the same way in your project.

IMPORTANT: don’t use any functions from the `random` module in any of your code.

IMPORTANT: you should only modify the methods that are listed as "must implement" below. Modifying any of the starter code will likely result in failing all of the tests.

To properly simulate blackjack, you will need to implement the following classes:

Player Class

Represents a participant in the game of blackjack.

Must meet the following requirements:

- Stores the individual cards that are dealt (i.e. cannot just store the sum of the cards, you need to track which specific cards you receive)

Provided methods:

- `decide_hit(self): dec`ides whether hit or stand by randomly selecting one of the two options.
    - Parameters:
        - `None`
    - Returns:
        - `True` to indicate a hit, `False` to indicate a stand

Must implement the following methods:

- `__init __ (self, name, dealer):` creates a new player with the given name, and initializes win,

  tie, and loss counters. Also creates an empty list to store the player's hand.
    - Parameters:
        - `name`: the name of the player
        - `dealer:` a dealer object from which the player can request cards
    - Returns:
        - `None`
- `deal_to(self, card_value):` used to deal a card to this player. The player must add the dealt card to his hand.
    - Parameters:
        - `card_value` is the numeric value of the received card
    - Returns:
        - `None`
- `play_round(self):` used to simulate this player playing a round of blackjack. Must use `decide_hit` to decide whether to hit or to stand. Calling the decide_hit function must be the very first thing done in this function. If the player decides to hit, continue play using

  `decide_hit` until the player either goes bust or the player decides to stand.
    - Parameters:
        - `None`
    - Returns:
        - `None`

- `discard_hand(self):` removes all the cards from the players hand. Used at the end of each round.
  - Parameters:
    - `None`
  - Returns:
    - `None`
- `card_sum(self)` (property method): gets the sum of the cards in the player's hand
  - Returns:
    - A number that represents the total value of all the cards in the player's hands
- `wins(self)` (property method): returns the number of wins for this player
  - Returns:
    - The number of wins recorded for this player
- `ties(self)` (property method): returns the number of ties for this player
  - Returns:
    - The number of ties recorded for this player
- `losses(self)` (property method): returns the number of losses for this player
  - Returns:
    - The number of losses recorded for this player
- `record_win(self):` adds a win to the internal win counter
  - Parameters:
    - `None`
  - Returns:
    - `None`
- `record_loss(self):` adds a loss to the internal loss counter
  - Parameters:
    - `None`
  - Returns:
    - `None`
- `record_tie(self):` adds a tie to the internal loss counter
  - Parameters:
    - `None`
  - Returns:
    - `None`

- `reset_stats(self):` sets all counters (win, loss, and tie) to zero
    - Parameters:
        - `None`
    - Returns:
        - `None`
- `__repr__(self):` returns a legible object representation including the player's name, current hand, and their wins, ties, and losses (use the format given in the doctests)
    - Parameters:
        - `None`
    - Returns:
        - `str`

CardDeck Class

Represents the deck of cards

Provided methods:

- `__repr__(self):` returns a legible representation of the deck of cards (simply outputs card values in order from the top of the deck to the bottom). Could be useful in debugging other parts of this assignment.
    - Parameters:
        - `None`
    - Returns:
        - `str`
- `shuffle(self):` shuffles the deck of cards. All cards that have previously been drawn with the `draw` method are returned to the deck.
    - Parameters:
        - `None`
    - Returns:
        - `None`

Must implement the following methods:

- `draw(self):` draws one card from the top of the deck and returns its value
    - Parameters:
        - `None`
    - Returns:

        - `int` the top of the deck, None if there is no top

Dealer Class

Represents the dealer in a game of blackjack. Must meet the following requirements:

- Inherits from the `Player` class, and always sets the name of the dealer object to "Dealer"

Must implement the following methods:

- `__init__(self):` creates a new dealer object. Should also initialize a new `CardDeck` object.

  Should pass some arbitrary string to the superclass's constructor as the dealer's name.
    - Parameters:
        - `None`
    - Returns:
        - `None`
- `shuffle_deck(self):` calls the `shuffle` method of the `CardDeck` object
    - Parameters:
        - `None`
    - Returns:
        - `None`
- `play_round(self):` used to simulate the dealer playing a round of blackjack. A dealer should hit as long as his total is below 17, and stands as soon as his total becomes 17 or higher
    - Parameters:
        - `None`
    - Returns:
        - `None`
- `signal_hit(self, player):` used by player objects to signal a "hit" (asking for another card). Draws a card from the deck and uses the `deal_to` method of the player class to deal the card.
    - Parameters:
        - `player:` the player object that is asking for another card
    - Returns:
        - `None`

BlackjackGame Class

The class which controls the blackjack simulation

Must implement the following methods:

- `__init__(self, player_names):` creates a new blackjack game. Should create new objects to

  represent the dealer and players in the game using the provided list. The player objects should be stored in a Python list, and the order of these objects should match the order of player_names.

  - Parameters:
    - `player_names:` list of names of players to create
  - Returns:
    - `None`

- `play_rounds(self, num_rounds=1):` simulates `num_rounds` rounds of play of a game. After all rounds have been played, returns a report on all rounds played. A single round of play involves the following steps: the dealer shuffles the deck, then deals one card to every player and then himself, then deals another card to every player and then himself, then each player plays through the round by standing/hitting. After the round is over, award wins/ties/losses to player as described in the
  Blackjack Overview. Wins/ties/losses do not have to be tracked for the dealer.

  - Parameters:
    - `num_rounds` (default = 1): the number of rounds to simulate
  - Returns:
    - A string listing the number of wins, ties, and losses for each player (Round number \n player name: hand, wins/ties/losses \n player name: hand, wins/ties/losses...)

- `reset_game(self):` resets the game to its original state (no players have cards in hand, and all stats are zero)

  - Parameters:
    - `None`
  - Returns:
    - `None`

Deliverables:

- Submit all 4 files using the original filenames (blackjack_game.py, card_deck.py, player.py, and dealer.py) in the Final Project Gradescope assignment before the due date