Blackjack

psp-9-8

1 Overview

In our previous lab we created objects representing playing cards and stacks of chips. In this lab we are going to implement a game of blackjack, using the objects we built previously. You and your partner will select a solution created in the Casino Night lab and import your old code. If both of your solutions met the required specification it shouldn't matter which you use. In a new file you will be creating a class to store a game of blackjack, and supporting functions. By the end of this lab you will have a simple text rendition of the game of Blackjack.

Rules

Here is an overview of our simplified blackjack rules:

- Face cards are worth 10 points. Aces are worth either 1 or 11 points, beneficial to the owner of the hand. Other cards are worth their rank value.
- The player is dealt two face up cards.
- The dealer is dealt a face-down card, and a face-up card.
- If both players have 21, the game ends in a tie.
- If either player or dealer have 21 and the other doesn't, that player wins the hand.
- The player may chose to hit as many times as they wish, or stand.
- If a player hits a new card is drawn. If that card makes their total hand value exceed 21, the player busts and loses.
- Once a player stands, the dealer reveals their face-down card and may choose to draw. The dealer will always draw at a 16 or lower, and if they go over 21, they bust.
- Once both players are done drawing, the highest hand wins.
- If the player wins, they double their money. If they lose, they get nothing, and if they tie, they get their wager back.

2 Learning Outcomes

By the end of this project students should be able to:

- read and write programs that utilize object-oriented programming;
- read and write programs with classes and be able to create objects;
- read and write programs that use method calls;
- work effectively with a partner using pair-programming;
- write an effective report that describes the students' problem solving process.

3 Pre-Lab Instructions

Do this part before coming to lab:

- Look through the rules of blackjack and walk step by step through the play of two different hands.
- Make sure to be as detailed as possible. Record when cards are shuffled, drawn, added to hands, where decisions are made, and what criteria those decisions are made from.
- Be prepared to show your work to the lab aide at the beginning of lab.

4 Lab Instructions

Do this part in lab:

4.1 BlackjackHand

First we will create a class for storing our hand in blackjack, so we can easily figure out how much our hand is worth, or display it to the screen

__init__()

A BlackjackHand doesn't take in any parameters, but it will need to set up a list for storing Cards.

add_card(new_card)

Adds a card to the hand.

parameters

- new_card - the Card object to be added to the hand

__str__()

Returns a string representation of the current hand. The resulting string should contain the __str__() value of each card, delimited by commas.

return

A string representing all the cards in the hand.
Example: "7 of Diamonds, King of Spades" or "Ace of Hearts, 8 of Clubs, Queen of Diamonds"

get_value()

Returns a numeric point value of the current hand. This should be the sum of the point values of each card. In our implementation of blackjack, an Ace should be either 11 points, or 1, to give the player the highest score without exceeding 21. Hint: First count the total points assuming all aces are 11, along with the number of aces, then decrement point total as the rules allow.

return

A numeric value representing the value of the hand.
Examples, respective to the examples in hand_str, above: 17 or 19

4.2 Blackjack

The Blackjack class is designed to represent a game of blackjack. This object can be thought of as a blackjack table, in which a series of hands are dealt from one or more decks of cards. **Read through all the methods before you start!** Below are specifications for the methods and instance variables that should be implemented in the Blackjack class:

bank

A instance variable should store the users ChipBank. This instance variable will be created durring __init__()

__init__(starting_dollars)

Creates a new Blackjack games in which the player starts with 'starting_dollars' worth of money. This method will need to set up instance variables to support the rest of the game. This should initialize the deck, and the player's chip stack. Remember to shuffle the deck after you create it.

parameters

- starting_dollars - Initial amount of money the player has to wager with.

Example: 250

draw()

This method draws and returns a card from the deck. If the deck is empty when this method is called, rebuild and reshuffle the deck. Make sure that all cards are drawn face up. Return a card object after it is removed from the deck.

return

A Card object removed from the deck.

start_hand(wager)

Starts a new hand of blackjack. Should initialize instance variables with a new hand for the player and the dealer. Remember that the first dealer's card should be set face down. This object should withdraw the wager amount from the ChipBank, and remember that value for the end of the game. Print both the player's hand and the dealer's hand to the player, and check to see if the player wins automatically with blackjack. If both the player AND the dealer have 21, it is a tie, or "push."

parameters

- wager - Numeric value representing the bet placed on the hand.

Example: 5

hit()

The player choses to hit. Draw a card for the player, and display the player's new hand. If they exceed 21 they bust, and if they get 21 exactly they are forced to stand. This method takes no parameters, and has no return value.

stand()

The player stands, and the dealer flips their hidden card face up and begins the process of drawing. The dealer only draws if their hands current value is 16 or less. If the dealer draws over 21, they bust and the player wins. After the dealer is done drawing, and neither has busted, the higher valued hand wins. Ties can occur, and are called "pushes." This method has no parameters and no return value.

end_hand(outcome)

This method should be called when a winner has been determined. This method is passed a parameter outcome, which will be "win", "lose", or "push" depending on if the player won, the dealer won, or their was a tie. If the player wins they should be given double their wager back. A push should refund the original wager, and a lose should result in no money being deposited. This method is also responsible for setting the two hands and the wager back to None. **This method should be called by any of the other methods in which a winner is found.**

parameters

- outcome - A string containing either "win", "lose", or "push".

game_active()

Returns True or False, indicating if there is a current hand in play. Should return True when start_hand() has been called, and end_hand() has not yet been called to close the game.

return

Returns a boolean value, True or False, depending on if there is an active hand that has not yet been resolved.

4.3 Testing

Here is given code to utilize the Blackjack methods

```
##### Sample Code #####
if __name__ == "__main__":
    blackjack = Blackjack(250)
    while blackjack.bank.value > 0:
        print("Your remaining chips: "+str(blackjack.bank))
        wager = int(input("How much would you like to wager? "))
        blackjack.start_hand(wager)
        while blackjack.game_active():
            choice = input("STAND or HIT: ").upper()
            if choice == "STAND": blackjack.stand()
            elif choice == "HIT": blackjack.hit()
        print()
    print("Out of money! The casino wins!")
```

Example Output

```
Your remaining chips: 2 blacks, 2 greens, 0 reds, 0 blues - totaling $250
How much would you like to wager? 10
Your starting hand: 9 of Spades, 10 of Clubs
dealers starting hand: <facedown>, 5 of Clubs
STAND or HIT: stand Dealer's hand: 3 of Clubs, 5 of Clubs
Dealer draws a 5 of Spades!
Dealer's hand is now 3 of Clubs, 5 of Clubs, 5 of Spades
Dealer draws a Ace of Hearts!
Dealer's hand is now 3 of Clubs, 5 of Clubs, 5 of Spades, Ace of Hearts
Dealer draws a 10 of Clubs!
Dealer's hand is now 3 of Clubs, 5 of Clubs, 5 of Spades, Ace of Hearts, 10 of Clubs
Dealer busts, you win!

Your remaining chips: 2 blacks, 2 greens, 2 reds, 0 blues - totaling $260
How much would you like to wager? 20
Your starting hand: 8 of Hearts, 7 of Clubs
dealers starting hand: <facedown>, 10 of Hearts STAND or HIT: hit
You drew a 3 of Hearts!
Your hand is now 8 of Hearts, 7 of Clubs, 3 of Hearts
STAND or HIT: stand
Dealer's hand: 10 of Hearts, 10 of Hearts
The dealer beats your hand!

Your remaining chips: 2 blacks, 1 greens, 3 reds, 0 blues - totaling $240
How much would you like to wager? 100
Your starting hand: 6 of Clubs, 4 of Spades
dealers starting hand: <facedown>, 9 of Hearts
STAND or HIT: hit You drew a 3 of Hearts!
Your hand is now 6 of Clubs, 4 of Spades, 3 of Hearts
STAND or HIT: hit You drew a 9 of Spades!
Your hand is now 6 of Clubs, 4 of Spades, 3 of Hearts, 9 of Spades
You bust!
Dealer's hand: 10 of Clubs, 9 of Hearts

Your remaining chips: 1 blacks, 1 greens, 3 reds, 0 blues - totaling $140
```

4.4 Extra Credit

Extra credit can be earned by creating a simulation mode for your blackjack. Modify the sample code above to include a menu, in which the user can choose to play blackjack like normal, or to simulate blackjack until the bank is empty. Make your simulation mode start a blackjack game with $1000 starting cash. The simulation should wager 5 dollars every hand. Like the dealer, the simulation should always hit on 16 or lower. Use the record() method on the chip bank

from the previous extra credit to record to a file during this simulation. After the simulation is over the program should print the number of hands before the chip stack is depleted, and the recorded file should be loaded into excel or another graphing program to generate a graph of the balance over time. Include the code modifications in your submission, and include the graph in the results section of your report.

4.5 Pep8

When you have completed the lab run pep8 against your code until all formatting errors have been corrected and your code is PEP 8 compliant. See the Getting Started lab if you need instructions on running the program, or the pep8 documentation found here.

5 Lab Report

Each pair of students will write a single lab report together and each student will turn in that same lab report on BBLearn. Submissions from each student on a pair should be identical.

Your lab report should begin with a preamble that contains:

- The lab assignment number and name
- Your name(s)
- The date
- The lab section

It should then be followed by four numbered sections:

1. Problem Statement

In this section you should describe the problem in **your** own words. The problem statement should answer questions like:

- What are the important features of the problem?
- What are the problem requirements?

This section should also include a reasonably complete list of requirements in the assignment. Following your description of the problem, include a bulleted list of specific features to implement. If there are any specific funtions, classes or numeric requirements given to you, they should be represented in this bulleted list.

2. Planning

In the second section you should describe what planning you did in order to solve the problem. You should include planning artifacts like sketches, diagrams, or pseudocode you may have used. You should also describe your planning process. List the specific data structures or techniques you plan on using, and why.

3. Implementation and Testing

In the third section you should describe how you implemented your plan. As directed by the lab instructor you should (as appropriate) include:

- a copy of your source code (Submitted in BBLearn as a .py file)
- a screen shot of your running application / solution
- results from testing

4. Reflection

In the last section you should reflect on the project. Consider different things you could have done to make your solution better. This might include code organization improvements, design improvements, etc.

You should also ask yourself what were the key insights or features of your solution? Were there alternative approaches or techniques you could have employed? How would these alternatives have impacted a different solution?

5. Partner Rating

Every assignment you are required to rate your partner with a score -1, 0 or 1. This should be submitted in the comment section of the BBLearn submission, and not in the report document. You do not have to tell your partner the rating you assign them. A rating of 1 indicates that your partner was particularly helpful or contributed exceptional effort. A rating of 0 indcates that your partner met the class expectations of them. Rating your partner at -1 means that they refused contribute to the project, failed to put in a resonable effort or actively blocked you from participating. If a student recieves three ratings of -1 they must attend a mandatory meeting with the instructor to dicuss the situation, and recieving additional -1 ratings beyond that, the student risks losing a letter grade, or even failing the course.

Colophon

This project was developed by Richard Lester and Dr. James Dean Palmer of Northern Arizona University.