DREXEL UNIVERSITY
Electrical and Computer Engineering
College of Engineering

ECEC 301
Programming for Engineers

Programming Assignment 3: Blackjack

Professor: James A. Shackleford

Section 062

Yonatan Carver
yac25@drexel.edu

Due Date: November 7, 2017

Yonatan Carver 11.8.2017
ECEC 301 – 062 Project #3

Introduction

The goal of this programming assignment is to create a Blackjack game using Python 2.7. By using programming techniques such as class inheritance, polymorphism, and various other aspects of object oriented programming, the programmer is able to create a playable card game.

Implementation Details

class Card(*object*)
class FaceCard(Card)
class AceCard(Card)

The `Card` class is the parent (or base) class that provides an underlying meaning for all playing cards in this game. Both the `FaceCard` and `AceCard` classes in the deck are child classes of the `Card` class. The `Card` class provides information about the following: the rank/value of cards (i.e. a soft and hard value between 2 and 10 for regular cards, a soft and hard value of 10 for all face cards, and a soft value of 11 and hard value of 1 for ace cards), the suite of the card (diamond, club, heart, spade), and the formatting for displaying the Unicode and name of the card (♠, ♥, ♦, and ♣).

The `FaceCard` class (function shown below) writes over both the `__init__` and `unicode` function of its parent class and inherits all other functions. Since the hard and soft rank values of every face card is ten, the initialization function of this class passes a value of "10" as the rank. This eliminates the need to re-state the `softValue` and `hardValue` functions. Although this class overwrites the rank stated in the parent class with the face value, the suit list remains defined as `suits = ['clubs', 'diamonds', 'hearts', 'spades']`. Additionally, the `unicode` function overwrites the previously stated value for rank with the value of each face card.

Likewise, since the Ace card is the only card that has two separate values for its rank (either a 1 or a 11), both the `softValue` and `hardValue` attributes must be updated (function shown below). The hard value of the card is defined in the `__init__` function. This eliminates the need for the redefinition of the `hardValue` function. As with `AceCard`'s sibling class, both the `softValue` and `unicode` functions are updated accordingly.

```
class FaceCard(Card):
    """
    Represents a Face Card (Jack, Queen, King) possessing a suit.  For face
    cards, both the softValue and hardValue are 10.
    """

    def __init__(self, face, suit):
        """ hard & soft vals of face cards is 10 """
        self._face = face                           # var for face (J, Q, K)
        super(FaceCard, self).__init__(10, suit)    # hard & soft val == 10

    def unicode(self):
        """
        Returns a Unicode string containing the Unicode symbol of the
        suit and the card rank
        """
        return "%s %s" % (self._suit, self._face)
```

```
class AceCard(Card):
    """
    Represents an Ace Card possessing a suit.  Aces have a softValue of 11
    and a hardValue of 1.
    """

    def __init__(self, suit):
        """ soft val = 11, hard val = 1 """
        super(AceCard, self).__init__(1, suit)    # hard val passed through init function

    def softValue(self):
        """ assign soft value = 11 """
        return 11

    def unicode(self):
        """ display unicode symbol for suit """
        return "%s %s" % (self._suit, 'A')
```

class CardContainer(object)
class Deck(CardContainer)
class Hand(CardContainer)

The CardContainer class is the parent class of both the Deck and Hand classes. The CardContainer class simply represents a container in which playing cards can be stored, moved around (shuffled), iterated through, added, and removed. Each one of these attributes is defined in their respective functions: __init__, __iter__, shuffle, popCard, pushCard. The __init__ function initializes an empty list (self._cards) that will eventually hold the 52 playing cards used in the game. The __iter__ function is a generator iterator that allows the user to iterate over the various cards in the deck. The shuffle function utilizes the function shuffle from python module "random" to randomly mix up the items in the self._cards list. Python has a built-in list method called "pop" that allows the programmer to remove a specific element in a list. In this script, popCard removes and returns the first card in the deck (element 0). The last function in this parent class is pushCard. pushCard uses another built-in list method called "append". Append adds a specified element to the end of a list – in this case, it adds a card to the empty list that was created in the __init_ function of this class.

While still maintaining certain aspects of a card container, the Deck child class further represents "card-iness". More specifically, this class represents the dealer's hand. The __init__ function calls the initialization function of its parent class with the line super(Deck, self).__init__(). This allows the Deck class to use the items declared in its super-class's initialization function (i.e. the creation of an empty deck) and add to its function without completely overwriting it. One unique feature of the _loadDeck function is that its name is preceded by an underscore. This is one of the many techniques programmers use to alert fellow programmers that this function should not be called outside of its class, i.e. it will only be used within the Deck class. This function populates the dealer's deck with the standard 52 playing cards using three for-loops. Each one of these for-loops iterates over a specific kind of card (numeric, face, and ace card) and populates the deck with the correct number of each card. The dealCard function deals a card so that it may be played in the game (and if there are no cards in the deck, this function re-populates and shuffles the deck so that a card may be played). One may notice that the _loadDeck function is called here but is not called anywhere else outside of this class.

The last child class of CardContainer is the Hand class. The Hand class represents the player's hand of cards. Similar to its sibling class, the Hand class inherits the initialization function from its parent class while also adding the option for a hole card (the card that only the player can see). Additionally, the following functions are implemented: getHoleCard, clear, getCard, getSoftScore, and getHardScore. getHoleCard simply displays the hole card in the player's hand. This function is called every time a round of Blackjack is played. clear removes all cards from self._cards (the deck) and the hole card in preparation for the next round of the game. The getCard function adds a card to the hand if a player decides to hit. If there are no cards in the player's hand, the getCard function puts the card in the hole. Likewise, if there are cards in the player's hand, the getCard function displays the card. The last two functions of this class are responsible for calculating and generating each player's score. As outlined earlier in the Card class, each card has both a soft and hard value (although only the Ace cards have different soft and hard values). Both the getSoftScore and getHardScore follow the same general format: initialize a score variable to 0, step through each card in the player's hand, add the soft or hard value to score, add the value of the hole card, return the soft or hard score. The getSoftScore and getHardScore functions are examples of polymorphic functions due to the fact that the inputs of these functions do not necessarily have to be of the same type. In this case, both the getSoftScore and getHardScore can accept any one of the types of cards (numeric, face, and ace) and return its rank value. The following snippet of code shows the polymorphic attributes of these functions.

```
    def getSoftScore(self):
        """
        Iterates through the Cards in the Hand and computes the soft
        score (i.e. Aces can be counted as 11 points).
        """
        score = 0       # initialize score to 0
        for card in self:       # for each card in hand...
            score += card.softValue()       # add soft value to score
        score += self._holeCard.softValue()     # add hole card value to score

        return score

    def getHardScore(self):
        """
        Iterates through the Cards in the Hand and computes the hard
        score.  Here, Aces are not counted as 11 points.
        """
        score = 0       # initialize score to 0
        for card in self:       # for each card in hand...
            score += card.hardValue()       # add hard value to score
        score += self._holeCard.hardValue()     # add hole card value to score

        return score
```

class Game(object)

The Game class holds the main gameplay instructions for the game that is Blackjack. The class can be broken down into functions that print out gameplay information to the terminal, adjust the player's or house's decks, provide the house with a specific strategy for playing the game, and check scores. The __init__ function initializes the deck, player hand, house hand, and shuffles the entire deck. The _printPlayer function displays information, in the terminal window, about the current status of the player's hand – the current score, current cards in the player's hand, current card in the player's hole, and the Unicode values for each card in the players' possession. The _revealHouseHand function is only called when a winner is determined as it displays the cards in the house's hand and in the houses' hole. Likewise, the _printHouse function is called to print out the cards in the house's hand (i.e., the cards that everyone can see and not the house's hole card). The _dealCardsToPlayer and _dealCardsToHouse functions removes a specific number of cards from the main deck and places them in either the player's or the house's hand respectively. The `_checkPlayerBust` and `_checkHouseBust` functions check to see if either the player or house has busted, that is, if their hard scores have exceeded a value of 21. If they have, these functions return a Boolean value of True and if they have not exceeded a value of 21, these functions return a Boolean value of False. The function `_doHouseAi` is essentially the function that plays against the player in the game. Its general strategy is as follows: if the house's score is under 18, the house will keep hitting. If the house gets a score of exactly 21, it will stop hitting. Also implemented in the `Game` class is a `_checkWhoWins` function that, surprisingly, checks to see who wins the game using the following rules:

-- If the Player goes over 21, the Player loses. (i.e. BUSTs)
-- If the House BUSTs (i.e. goes over 21), the Player wins.
-- If the Player gets exactly 21, the Player wins.
-- If the House gets exactly 21, the Player loses.
-- If both the House and the Player get 21, the Player wins.
-- If nobody BUSTs or has 21, whoever has the larger score wins.
-- If the scores are tied, the Player wins.

Additionally, the `_prompChoices` function allows the programmer to quickly implement choice questions with user input. The programmer provides this function with a prompt/question and choices that the user can choose from. The function `_startNewHand` clears both the House's and Player's hand and gives them two new cards. This function is called at the beginning of the game. The function `playHand` is the function that essentially loops through the steps necessary to run Blackjack.

```python
def playHand(self):
        """
        Plays a new hand. The general procedure follows:
          [1] Starts a new hand
          [2] Ask the user to hit or stay until they choose to stay
          [3] Execute the House AI
          [4] Figure out who won and display the winner
          [5] Ask the player if they want to play another hand

        Naturally, events such as a BUST (i.e. taking a hit and going over
        21) will prematurely end the hand.
        """
        self._startNewHand()

        self._printHouse()

        while True:
            self._printPlayer()
            options = self._promptChoices('[h]it or [s]tay: ', ['h', 's'])
            if options is 'h':
                self._dealCardsToPlayer(1)
                if self._checkPlayerBust():
                    break
            else:
                break

        self._doHouseAi()

        self._checkWhoWins()

        choice = self._promptChoices('Play Again? [y]es, [n]o: ', ['y', 'n'])
        if choice is 'y':
            return True
        elif choice is 'n':
            return False
```

def main()

This main function assigns a variable (G) to the class Game and initiates the playHand function, as described above, that runs the game of Blackjack. If the user breaks out of the loop, this function returns a goodbye string and the game is exited.

As with many Python programs, it is typically good practice to initiate a script by saying the following:

```
if __name__ == '__main__':
    main()
```

In Python, '__main__' is the name of the scope in which top-level code is executed. Every module in Python has an attribute called __name__, which is set to __main__ when a module is run as a main program. This snippet of code runs Blackjack.

How to Run and Play

The game-play layout is relatively straightforward. To initiate the game, the user must type "python blackjack.py" in the terminal window. Given that Python 2.7 is installed on the host machine, the game will launch. The game will show the House and Player Cards and will prompt the user if they would like to hit or stay. By inputting an "h" the Player will receive another card and by inputting a "s" the Player will not gain a card and their score will remain unchanged. The goal is to get the total score equal to or as close to 21 as possible without going over. The House will also try to get a score of 21. If the Player chooses to hit and their total score exceeds 21, the Player busts and loses. Likewise, if the Player hits and gets a score close to 21 or equal to 21 and the House does not, the Player wins. Depending on whether the Player or the House busts or wins, the hole and hand cards will be displayed and a prompt will ask the Player if they would like to play again. If the Player chooses a "y" to play again, the Blackjack game will repeat, but if the Player chooses an "n" the game will exit.

Yonatan Carver 11.8.2017
ECEC 301 – 062 Project #3

Gameplay Examples

Example 1

```
BLACKJACK GAME                       COMMENTS
python blackjack.py                  Initialize the blackjack game

House:
  Showing: ♣ K,                      House is showing a King
Player: [18, 18]                     Player has a hand of total 18
     Hole: ♥ 8                       Card in Player’s hole
  Showing: ♠ J,                      Player is showing a Jack

[h]it or [s]tay: s                   User prompt to hit or stay (stay)
-------------------------------
House Had: [13, 13]                  House had a total score of 13
  In The Hole: ♠ 3                   House had a 3 in the hole
  Showing: ♣ K,                      House had a King
-------------------------------
You win!                             Player won!

Play Again? [y]es, [n]o: y           User prompt to play Blackjack again (yes)
House:
  Showing: ♣ 2,                      House is showing a 2
Player: [10, 10]                     Player has a hand of total 10
     Hole: ♥ 4                       Card in Player’s hole
  Showing: ♠ 6,                      Player is showing a 6
[h]it or [s]tay: h                   User prompt to hit or stay (hit)
Player: [20, 20]                     Player has a hand of total 20 now
     Hole: ♥ 4                       Card in Player’s hole
  Showing: ♠ 6, ♦ J,                 Player is showing a 6 and a 3

[h]it or [s]tay: s                   User prompt to hit or stay (stay)
-------------------------------
House Had: [4, 4]                    House had a total score of 4
  In The Hole: ♠ 2                   House had a 2 in the hole
  Showing: ♣ 2,                      House is showing a 2
-------------------------------
You win!                             Player won!

Play Again? [y]es, [n]o: n           User prompt to play again (no)

Bye Bye!                             See ya!
```

Yonatan Carver   11.8.2017
ECEC 301 – 062   Project #3

Example 2

```
BLACKJACK GAME                       COMMENTS
House:
  Showing: ♣ 10,                     House is showing a 10
Player: [11, 11]                     Player has a hand of total 11
     Hole: ♣ 7                       Card in Player’s hole
  Showing: ♣ 4,                      Player is showing a 4

[h]it or [s]tay: h                   User prompt to hit or stay (hit)
Player: [14, 14]                     Player has a hand of total 14
     Hole: ♣ 7                       Card in Player’s hole
  Showing: ♣ 4, ♠ 3,                 Player is showing a 4 and a 3

[h]it or [s]tay: h                   User prompt to hit or stay (hit)
-------------------------------
House Had: [24, 24]                  House had a total score of 24
  In The Hole: ♠ 2                   House had a 2 in the hole
  Showing: ♣ 10, ♣ 6, ♦ 6,           House is showing a 10, 6, and 6
-------------------------------
You busted, you lose                 Player busted, Player lost

Play Again? [y]es, [n]o: y           User prompt to play again (yes)
House:
  Showing: ♥ 10,
Player: [14, 14]                     Player has a hand of total 14
     Hole: ♦ J                       Card in Player’s hole
  Showing: ♦ 4,                      Player is showing a 4 and a 4

[h]it or [s]tay: h                   User prompt to hit or stay (hit)
-------------------------------
House Had: [11, 21]                  House had a hard score of 11 and a soft score
                                     of 21
  In The Hole: ♣ A                   House had an Ace in the hole
  Showing: ♥ 10,                     House is showing a 10
-------------------------------
You busted, you lose                 Player busted, Player lost

Play Again? [y]es, [n]o: n           User prompt to play again (no)

Bye Bye!                             Adios!
```