Milestone Project 2 - Walkthrough Steps Workbook

Below is a set of steps for you to follow to try to create the Blackjack Milestone Project game!

Game Play

To play a hand of Blackjack the following steps must be followed:

1. Create a deck of 52 cards
2. Shuffle the deck
3. Ask the Player for their bet
4. Make sure that the Player's bet does not exceed their available chips
5. Deal two cards to the Dealer and two cards to the Player
6. Show only one of the Dealer's cards, the other remains hidden
7. Show both of the Player's cards
8. Ask the Player if they wish to Hit, and take another card
9. If the Player's hand doesn't Bust (go over 21), ask if they'd like to Hit again.
10. If a Player Stands, play the Dealer's hand. The dealer will always Hit until the Dealer's value meets or exceeds 17
11. Determine the winner and adjust the Player's chips accordingly
12. Ask the Player if they'd like to play again

Playing Cards

A standard deck of playing cards has four suits (Hearts, Diamonds, Spades and Clubs) and thirteen ranks (2 through 10, then the face cards Jack, Queen, King and Ace) for a total of 52 cards per deck. Jacks, Queens and Kings all have a rank of 10. Aces have a rank of either 11 or 1 as needed to reach 21 without busting. As a starting point in your program, you may want to assign variables to store a list of suits, ranks, and then use a dictionary to map ranks to values.

The Game

Imports and Global Variables

Step 1: Import the random module. This will be used to shuffle the deck prior to dealing. Then, declare variables to store suits, ranks and values. You can develop your own system, or copy ours below. Finally, declare a Boolean value to be used to control `while` loops. This is a common practice used to control the flow of the game.

```
suits = ('Hearts', 'Diamonds', 'Spades', 'Clubs')
ranks = ('Two', 'Three', 'Four', 'Five', 'Six', 'Seven', 'Eight', 'Nine',
 'Ten', 'Jack', 'Queen', 'King', 'Ace')
values = {'Two':2, 'Three':3, 'Four':4, 'Five':5, 'Six':6, 'Seven':7, 'Ei
ght':8, 'Nine':9, 'Ten':10, 'Jack':10,
         'Queen':10, 'King':10, 'Ace':11}
```

In [1]:

```
import random

suits = ('Hearts', 'Diamonds', 'Spades', 'Clubs')
ranks = ('Two', 'Three', 'Four', 'Five', 'Six', 'Seven', 'Eight', 'Nine', 'Ten',
 'Jack', 'Queen', 'King', 'Ace')
values = {'Two':2, 'Three':3, 'Four':4, 'Five':5, 'Six':6, 'Seven':7, 'Eight':8,
 'Nine':9, 'Ten':10, 'Jack':10,
         'Queen':10, 'King':10, 'Ace':11}

playing = True
```

Class Definitions

Consider making a Card class where each Card object has a suit and a rank, then a Deck class to hold all 52 Card objects, and can be shuffled, and finally a Hand class that holds those Cards that have been dealt to each player from the Deck.

Step 2: Create a Card Class
A Card object really only needs two attributes: suit and rank. You might add an attribute for "value" - we chose to handle value later when developing our Hand class.
In addition to the Card's __init__ method, consider adding a __str__ method that, when asked to print a Card, returns a string in the form "Two of Hearts"

In [2]:

```
class Card():

    def __init__(self,suit,rank):
        self.suit = suit
        self.rank = rank

    def __str__(self):
        return self.rank+ " of "+self.suit
```

Step 3: Create a Deck Class

Here we might store 52 card objects in a list that can later be shuffled. First, though, we need to *instantiate* all 52 unique card objects and add them to our list. So long as the Card class definition appears in our code, we can build Card objects inside our Deck __init__ method. Consider iterating over sequences of suits and ranks to build out each card. This might appear inside a Deck class __init__ method:

```
for suit in suits:
    for rank in ranks:
```

In addition to an __init__ method we'll want to add methods to shuffle our deck, and to deal out cards during gameplay.

OPTIONAL: We may never need to print the contents of the deck during gameplay, but having the ability to see the cards inside it may help troubleshoot any problems that occur during development. With this in mind, consider adding a __str__ method to the class definition.

In [3]:

```
class Deck():

    def __init__(self):
        self.deck = []  # start with an empty list
        for suit in suits:
            for rank in ranks:
                self.deck.append(Card(suit,rank))

    def __str__(self):
        deck_comp = ''
        for card in self.deck:
            deck_comp += '\n'+ card.__str__()
        return "The deck has: "+ deck_comp

    def shuffle(self):
        random.shuffle(self.deck)

    def deal(self):
        single_card = self.deck.pop()
        return single_card
```

TESTING: Just to see that everything works so far, let's see what our Deck looks like!

In [4]:

```
test_deck = Deck()
test_deck.shuffle()
print(test_deck)
```

```
The deck has:
King of Clubs
King of Diamonds
Three of Clubs
Ten of Diamonds
Five of Hearts
Eight of Spades
Four of Clubs
Nine of Clubs
Queen of Hearts
Jack of Hearts
Three of Diamonds
Six of Hearts
Queen of Clubs
Ten of Hearts
Nine of Hearts
Five of Diamonds
Five of Spades
Ace of Clubs
Three of Spades
Ace of Spades
Four of Spades
Ace of Hearts
Two of Clubs
Four of Diamonds
Six of Spades
Eight of Diamonds
Queen of Diamonds
Four of Hearts
Two of Hearts
Seven of Diamonds
Three of Hearts
Seven of Spades
Jack of Spades
Eight of Clubs
Nine of Spades
King of Hearts
Ten of Clubs
Jack of Diamonds
Two of Spades
Five of Clubs
King of Spades
Two of Diamonds
Seven of Clubs
Queen of Spades
Nine of Diamonds
Six of Diamonds
Ten of Spades
Jack of Clubs
Seven of Hearts
Six of Clubs
Ace of Diamonds
Eight of Hearts
```

Great! Now let's move on to our Hand class.

Step 4: Create a Hand Class

In addition to holding Card objects dealt from the Deck, the Hand class may be used to calculate the value of those cards using the values dictionary defined above. It may also need to adjust for the value of Aces when appropriate.

In [5]:

```
class Hand():
    def __init__(self):
        self.cards = []  # start with an empty list as we did in the Deck class
        self.value = 0   # start with zero value
        self.aces = 0    # add an attribute to keep track of aces

    def add_card(self,card):
        self.cards.append(card)
        self.value += values[card.rank]
        if card.rank == 'Ace':
            self.aces += 1

    def adjust_for_ace(self):
        while self.value > 21 and self.aces:
            self.value -= 10
            self.aces -= 1
```

In [6]:

```
test_deck = Deck()
test_deck.shuffle()

test_player = Hand()
pulled_card = test_deck.deal()
print(pulled_card)
test_player.add_card(pulled_card)
print(test_player.value)
```

```
Seven of Hearts
7
```

In [ ]:

```

```

Step 5: Create a Chips Class

In addition to decks of cards and hands, we need to keep track of a Player's starting chips, bets, and ongoing winnings. This could be done using global variables, but in the spirit of object oriented programming, let's make a Chips class instead!

In [7]:

```
class Chips():

    def __init__(self,total=100):
        self.total = total  # This can be set to a default value or supplied by
 a user input
        self.bet = 0

    def win_bet(self):
        self.total += self.bet

    def lose_bet(self):
        self.total -= self.bet
```

Function Defintions

A lot of steps are going to be repetitive. That's where functions come in! The following steps are guidelines - add or remove functions as needed in your own program.

Step 6: Write a function for taking bets

Since we're asking the user for an integer value, this would be a good place to use `try` / `except` . Remember to check that a Player's bet can be covered by their available chips.

In [8]:

```
def take_bet(chips):
    while True:
        try:
            chips.bet = int(input("How many chips would you like to bet ?"))
        except:
            print("Sorry please provide an integer")
        else:
            if chips.bet > chips.total:
                print("Sorry, you do not have enough chips! You have {}".format(
chips.total))
            else:
                break
```

Step 7: Write a function for taking hits

Either player can take hits until they bust. This function will be called during gameplay anytime a Player requests a hit, or a Dealer's hand is less than 17. It should take in Deck and Hand objects as arguments, and deal one card off the deck and add it to the Hand. You may want it to check for aces in the event that a player's hand exceeds 21.

In [9]:

```
def hit(deck,hand):
    single_card = deck.deal()
    hand.add_card(single_card)
    hand.adjust_for_ace()
```

Step 8: Write a function prompting the Player to Hit or Stand

This function should accept the deck and the player's hand as arguments, and assign playing as a global variable.

If the Player Hits, employ the hit() function above. If the Player Stands, set the playing variable to False - this will control the behavior of a `while` loop later on in our code.

In [10]:

```
def hit_or_stand(deck,hand):
    global playing  # to control an upcoming while loop

    while True:
        x = input('Hit or Stand ? Enter h or s :')
        if x[0].lower() == 'h':
            hit(deck,hand)
        elif x[0].lower() == 's':
            print("Player stands, Dealer's Turn")
            playing = False
        else:
            print("Sorry, I didn't understand that, Please enter h or s only!")
            continue
        break
```

Step 9: Write functions to display cards

When the game starts, and after each time Player takes a card, the dealer's first card is hidden and all of Player's cards are visible. At the end of the hand all cards are shown, and you may want to show each hand's total value. Write a function for each of these scenarios.

In [11]:

```
def show_some(player,dealer):
    print('Dealers Hand : ')
    print('one card hidden ! ')
    print(dealer.cards[1])
    print('\n')
    print('Players Hand : ')
    for card in player.cards:
        print(card)

def show_all(player,dealer):
    print('Dealers Hand : ')
    for card in dealer.cards:
        print(card)
    print('\n')
    print('Players Hand : ')
    for card in player.cards:
        print(card)
```

Step 10: Write functions to handle end of game scenarios

Remember to pass player's hand, dealer's hand and chips as needed.

In [12]:

```
def player_busts(player,dealer,chips):
    print('BUST Player !')
    chips.lose_bet()

def player_wins(player,dealer,chips):
    print('Player WINS !')
    chips.win_bet()

def dealer_busts(player,dealer,chips):
    print('Player WINS ! Dealer BUSTED !')
    chips.win_bet()

def dealer_wins(player,dealer,chips):
    print('Dealer WINS !')
    chips.lose_bet()

def push():
    print('Dealer and player tie ! PUSH')
```

And now on to the game!!

In [13]:

```
while True:
    # Print an opening statement
    print("Welcome to BlackJack")
    # Create & shuffle the deck, deal two cards to each player
    deck = Deck()
    deck.shuffle()

    player_hand = Hand()
    player_hand.add_card(deck.deal())
    player_hand.add_card(deck.deal())

    dealer_hand = Hand()
    dealer_hand.add_card(deck.deal())
    dealer_hand.add_card(deck.deal())

    # Set up the Player's chips
    player_chips = Chips()

    # Prompt the Player for their bet
    take_bet(player_chips)

    # Show cards (but keep one dealer card hidden)
    show_some(player_hand,dealer_hand)

    while playing:  # recall this variable from our hit_or_stand function
        # Prompt for Player to Hit or Stand
        hit_or_stand(deck,player_hand)

        # Show cards (but keep one dealer card hidden)
        show_some(player_hand,dealer_hand)

        # If player's hand exceeds 21, run player_busts() and break out of loop
        if player_hand.value > 21:
            player_busts(player_hand,dealer_hand,player_chips)
            break

    # If Player hasn't busted, play Dealer's hand until Dealer reaches 17
    if player_hand.value <= 21:
        while dealer_hand.value < 17:
            hit(deck,dealer_hand)

        # Show all cards
        show_all(player_hand,dealer_hand)

        # Run different winning scenarios
        if dealer_hand.value >21:
            dealer_busts(player_hand,dealer_hand,player_chips)
        elif dealer_hand.value > player_hand.value:
            dealer_wins(player_hand,dealer_hand,player_chips)
        elif dealer_hand.value < player_hand.value:
            player_wins(player_hand,dealer_hand,player_chips)
        else:
            push(player_hand,dealer_hand)

    # Inform Player of their chips total
    print('\n Player total chips are at : {}'.format(player_chips.total))
    # Ask to play again
    new_game = input("Would you like to play another hand ? y/n")
```

```
    if new_game[0].lower() == 'y':
        playing = True
        continue
    else:
        print('Thank you for playing !')
        break
```

```
Welcome to BlackJack
How many chips would you like to bet ?50
Dealers Hand :
one card hidden !
Nine of Clubs


Players Hand :
Five of Diamonds
Nine of Hearts
Hit or Stand ? Enter h or s :h
Dealers Hand :
one card hidden !
Nine of Clubs


Players Hand :
Five of Diamonds
Nine of Hearts
Six of Hearts
Hit or Stand ? Enter h or s :s
Player stands, Dealer's Turn
Dealers Hand :
one card hidden !
Nine of Clubs


Players Hand :
Five of Diamonds
Nine of Hearts
Six of Hearts
Dealers Hand :
Seven of Spades
Nine of Clubs
Ace of Spades


Players Hand :
Five of Diamonds
Nine of Hearts
Six of Hearts
Player WINS !

 Player total chips are at : 150
Would you like to play another hand ? y/nn
Thank you for playing !
```

And that's it! Remember, these steps may differ significantly from your own solution. That's OK! Keep working on different sections of your program until you get the desired results. It takes a lot of time and patience! As always, feel free to post questions and comments to the QA Forums.

Good job!

In [ ]: