Milestone Project 2 - Solution

Below is my implementation of a simple game of Blackjack. Notice the use of OOP and classes for the cards and hands.

Let's start by defining some global objects for the cards, tuples and a dict.

In [2]:
```
# Used for card shuffle
import random

# Boolean used to know if hand is in play
playing = False

chip_pool = 100 # Could also make this a raw input.

bet = 1

restart_phrase = "Press 'd' to deal the cards again, or press 'q' to quit"
```

In [3]:
```
# Hearts, Diamonds,Clubs,Spades
suits = ('H','D','C','S')

# Possible card ranks
ranking = ('A', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q', 'K')

# Point values dict (Note: Aces can also be 11, check self.ace for details)
card_val = {'A':1, '2':2, '3':3, '4':4, '5':5, '6':6, '7':7, '8':8, '9':9, '1
0':10, 'J':10, 'Q':10, 'K':10}
```

Now I'll make a card class, it will have some basic ID functions, and then some functions to grab the suit and rank of the card.

```
In [4]: # Create a card class
        class Card:

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

            def __str__(self):
                return self.suit + self.rank

            def grab_suit(self):
                return self.suit

            def grab_rank(self):
                return self.rank

            def draw(self):
                print (self.suit + self.rank)
```

Now I'll make a hand class, this class will have functions to take into account the Ace situation

In [5]:
```
# Create a hand class
class Hand:

    def __init__(self):
        self.cards = []
        self.value = 0
        # Aces can be 1 or 11 so need to define it here
        self.ace = False

    def __str__(self):
        ''' Return a string of current hand composition'''
        hand_comp = ""

        # Better way to do this? List comprehension?
        for card in self.cards:
            card_name = card.__str__()
            hand_comp += " " + card_name

        return 'The hand has %s' %hand_comp

    def card_add(self,card):
        ''' Add another card to the hand'''
        self.cards.append(card)

        # Check for Aces
        if card.rank == 'A':
            self.ace = True
        self.value += card_val[card.rank]

    def calc_val(self):
        '''Calculate the value of the hand, make aces an 11 if they don't bust
 the hand'''
        if (self.ace == True and self.value < 12):
            return self.value + 10
        else:
            return self.value

    def draw(self,hidden):
        if hidden == True and playing == True:
            #Don't show first hidden card
            starting_card = 1
        else:
            starting_card = 0
        for x in range(starting_card,len(self.cards)):
            self.cards[x].draw()
```

Next I'll make a deck class

In [6]:
```
class Deck:

    def __init__(self):
        ''' Create a deck in order '''
        self.deck = []
        for suit in suits:
            for rank in ranking:
                self.deck.append(Card(suit,rank))

    def shuffle(self):
        ''' Shuffle the deck, python actually already has a shuffle method in
 its random lib '''
        random.shuffle(self.deck)

    def deal(self):
        ''' Grab the first item in the deck '''
        single_card = self.deck.pop()
        return single_card

    def __str__(self):
        deck_comp = ""
        for card in self.cards:
            deck_comp += " " + deck_comp.__str__()

        return "The deck has" + deck_comp
```

Now that the classes are done, time for the cool part, creating the actual game!

First off, making a bet. Need to check if the bet amount is within the integer.

In [7]:
```
# First Bet
def make_bet():
    ''' Ask the player for the bet amount and '''

    global bet
    bet = 0

    print ' What amount of chips would you like to bet? (Enter whole integer p
lease) '

    # While loop to keep asking for the bet
    while bet == 0:
        bet_comp = raw_input() # Use bet_comp as a checker
        bet_comp = int(bet_comp)
        # Check to make sure the bet is within the remaining amount of chips l
eft.
        if bet_comp >= 1 and bet_comp <= chip_pool:
            bet = bet_comp
        else:
            print "Invalid bet, you only have " + str(chip_pool) + " remainin
g"
```

Next, make a function setting up the game and for dealing out the cards.

In [8]:
```
def deal_cards():
    ''' This function deals out cards and sets up round '''

    # Set up all global variables
    global result,playing,deck,player_hand,dealer_hand,chip_pool,bet

    # Create a deck
    deck = Deck()

    #Shuffle it
    deck.shuffle()

    #Set up bet
    make_bet()

    # Set up both player and dealer hands
    player_hand = Hand()
    dealer_hand = Hand()

    # Deal out initial cards
    player_hand.card_add(deck.deal())
    player_hand.card_add(deck.deal())

    dealer_hand.card_add(deck.deal())
    dealer_hand.card_add(deck.deal())

    result = "Hit or Stand? Press either h or s: "

    if playing == True:
        print 'Fold, Sorry'
        chip_pool -= bet

    # Set up to know currently playing hand
    playing = True
    game_step()
```

Now make the hit function

In [9]:
```
def hit():

    ''' Implement the hit button '''
    global playing,chip_pool,deck,player_hand,dealer_hand,result,bet

    # If hand is in play add card
    if playing:
        if player_hand.calc_val() <= 21:
            player_hand.card_add(deck.deal())

        print "Player hand is %s" %player_hand

        if player_hand.calc_val() > 21:
            result = 'Busted! '+ restart_phrase

            chip_pool -= bet
            playing = False

    else:
        result = "Sorry, can't hit" + restart_phrase

    game_step()
```

Now make the stand function

In [10]:
```
def stand():
    global playing,chip_pool,deck,player_hand,dealer_hand,result,bet
    ''' This function will now play the dealers hand, since stand was chosen
 '''

    if playing == False:
        if player_hand.calc_val() > 0:
            result = "Sorry, you can't stand!"

    # Now go through all the other possible options
    else:

        # Soft 17 rule
        while dealer_hand.calc_val() < 17:
            dealer_hand.card_add(deck.deal())

        # Dealer Busts
        if dealer_hand.calc_val() > 21:
            result = 'Dealer busts! You win!' + restart_phrase
            chip_pool += bet
            playing = False

        #Player has better hand than dealer
        elif dealer_hand.calc_val() < player_hand.calc_val():
            result = 'You beat the dealer, you win!' + restart_phrase
            chip_pool += bet
            playing = False

        # Push
        elif dealer_hand.calc_val() == player_hand.calc_val():
            result = 'Tied up, push!' + restart_phrase
            playing = False

        # Dealer beats player
        else:
            result = 'Dealer Wins!' + restart_phrase
            chip_pool -= bet
            playing = False
    game_step()
```

Function to print results and ask user for next step

```
In [11]: def game_step():
             'Function to print game step/status on output'

             #Display Player Hand
             print ""
             print('Player Hand is: '),
             player_hand.draw(hidden =False)

             print 'Player hand total is: '+str(player_hand.calc_val())

             #Display Dealer Hand
             print('Dealer Hand is: '),
             dealer_hand.draw(hidden=True)

             # If game round is over
             if playing == False:
                 print  " --- for a total of " + str(dealer_hand.calc_val() )
                 print "Chip Total: " + str(chip_pool)
             # Otherwise, don't know the second card yet
             else:
                 print " with another card hidden upside down"

             # Print result of hit or stand.
             print result

             player_input()
```

Function for exiting the game

```
In [12]: def game_exit():
             print 'Thanks for playing!'
             exit()
```

Function to read user input

```
In [13]: def player_input():
             ''' Read user input, lower case it just to be safe'''
             plin = raw_input().lower()


             if plin == 'h':
                 hit()
             elif plin == 's':
                 stand()
             elif plin == 'd':
                 deal_cards()
             elif plin == 'q':
                 game_exit()
             else:
                 print "Invalid Input...Enter h, s, d, or q: "
                 player_input()
```

Make a quick intro for the game

In [14]:
```
def intro():
    statement = '''Welcome to BlackJack! Get as close to 21 as you can without
 going over!
    Dealer hits until she reaches 17. Aces count as 1 or 11.
    Card output goes a letter followed by a number of face notation'''
    print statement
```

Now to play the game!

In [ ]:
```
'''The following code will initiate the game! (Note: Need to Run all Cells)'''

# Create a Deck
deck = Deck()
#Shuffle it
deck.shuffle()
# Create player and dealer hands
player_hand = Hand()
dealer_hand = Hand()
#Print the intro
intro()
# Deal out the cards and start the game!
deal_cards()
```