**1. Card class:**

Class to represent a playing card

1. Attributes:

**suit**: a public attribute representing suit of the card. Possible values:  
“Spades”, “Hearts”, “Diamonds” and “Clubs”

**rank**: a public attribute representing rank of the card. Possible values: "Ace",  
"2", "3", "4", "5", "6", "7", "8", "9", "10", "Jack", "Queen", "King".

1. Properties:

**value**: A public read-only property

represents the point value of the card

It is only dependent on the rank of the card:

An Ace is worth 11 points,  
Jack/Queen/King each is 10 points

the rest of the cards are worth their rank value

Hint: You can use the int function to convert the rank of the non-face cards to  
the corresponding point value.

c. A constructor:

This takes two required arguments:

first one is **rank**

second one is **suit**  
 **initializes** the corresponding public attributes

make sure the **order** of the arguments is **maintained** –

the first one is the rank and the second one is the suit.

Card(“King”, “Hearts”) will create a card object representing “King of Hearts”

Card(“7”, “Spades”) will create a card object representing “7 of Spades”

d. Methods

**displayCard**: a method that takes no parameters

returns a string of the form “rank of suit suitsymbol”

E.g., “King of Hearts ♥”, “10 of Diamonds ♦", “3 of Clubs ♣”

Hint: For your convenience a dictionary is provided at the top of the object.py file that has these symbols corresponding to each suit. Use this dictionary here.

**2. Deck class:**

Class to represent a deck of cards

a. Attributes:

**\_\_deck:** a private attribute which is a list of objects of Card class.

1. Properties:

**count:** A read-only property

equal to the count of cards currently in the\_\_deck list.

1. A constructor:  
    This takes **no arguments**

It **initializes \_\_deck attribute** *with 52 objects* of Card class with

one of each card type:

13 with suit = “Spades” | ranks from “2” up to “10”, Ace/Jack/Queen/King

13 with suit = “Hearts” | ranks from “2” up to “10”, Ace/Jack/Queen/King

13 with suit = “Diamonds” | ranks from “2” up to “10”, Ace/Jack/Queen/King

13 with suit = “Clubs” | ranks from “2” up to “10”, Ace/Jack/Queen/King

so that \_\_deck has all unique 52 cards

These can be in perfect order.

There are many ways to fill in this list of 52, using loop constructs.

Hint:

One possible way is to use two lists, one initialized to 4 possible values  
of suit and other initialized to 13 possible values of rank.

Then use a nested loop to loop over these two creating all possible pairings of suit and rank to  
create the 52 cards.

But feel free to use any other way to use loops here.

1. Methods  
    **shuffle**: a method with no arguments that shuffles the deck to a random  
   order.

Hint: Use the shuffle method of the random module that takes a list and shuffles in place.

**dealCard:**

a method with no arguments

removes a card from the deck

and returns it

It should return None if no cards left in the deck

**3. Hand class**

Class to represent the dealer’s hand or the player’s hand

a. Attributes:

**\_\_cards**: a private attribute which is a list of objects of Card class.

b. Properties:

**count**: A read-only property equal to the count of cards currently in the  
\_\_cards list.

**points**: A read-only property equal to total points of the cards in the hand.

c. A constructor:

This takes **no arguments.**

It **initializes \_\_cards** attribute to an empty list.

d. Methods:

**addCard**: a method that takes one argument of Card class object and  
appends it to the \_\_cards list.

**displayHand**:

a method with no arguments

prints the hand by printing displayCard method [of each card in the \_\_cards list.]