Final Guide for Presentation

THIS GUIDE ASSUMES THAT THE USER HAS ALREADY COMPLETED THE 'DECK OF CARDS IN OBJECT ORIENTED PROGRAMMING' TUTORIAL.

PART 1 Adding values to the deck of cards

The deck of cards in the previous project was a list of string values for the cards. These steps will show how to make some simple changes to the code that allow the cards to be given values can be used in card games.

1:
The first thing we need to do is consider when the cards will be assigned values. Since you have already created an __init__ for Card instances when they are created, it makes sense to do it then.

Go to your Card classes __init__ function:

```
#initalised with the card
def __init__(self, suit, number):
    self._suit = suit
    self._number = number
```

2:

In order to assign the card instances in your deck a value, we will create conditions against the string value they currently have already. This will allow us to create separate values for the numbered and pictured cards.

We will start with the numbered cards by using the list comprehension example from the last project:

```
#initalised with the card
def __init__(self, suit, number):
    self._suit = suit
    self._number = number
    if self._number in [str(n) for n in range (2,11)]:
        self._value = int(number)
```

The If statement here works by checking the self._number value created when the card was created. If it's within the str version of every n in the range of 2 up to but not including 11, it will then change the self._value value number to an integer version of the string. Because the values in the list are 2 through 10, we can convert them to numbers

3:

We will then create the values for picture cards. The Jack, Queen and King are all worth 10, so we add an Elif statement. This runs if the first If statement isn't true, which will be the case for the pictured cards.
This Elif statement checks to see if the self._numbervalue is in the list described. If it is, then the card's assigned value is 10.

```
if self._number in [str(n) for n in range (2,11)]:
    self._value = int(number)
elif self._number in ["Jack","Queen","King"]:
    self._value = 10
```

We have one more card - the Ace. Because the Ace is valued as 1, We will add one more Elif statement for it:

```
        elif self.number in ["Ace"]:
            self._value = 1
```

PART 2

SHUFFLING THE DECK

1: Remove the print (self.cards) statement on the Deck's __init__ function - we know now that the cards exist - and do not need the whole deck printing out each time the program is run.

2: We will now instead create a function to show the cards before and after they have been shuffled.

```
def ShowTheDeck():
    print (f"The deck has: {str(mydeck._cards)} \n")
```

This function shows all of the cards in the deck. The reason we have added this as a separate function is so that it we can show the cards more than once, to check whether our shuffle function has worked.

3: We Will now create a function that takes all of the cards in the deck and shuffles them.

```
83  # Shuffling the Deck
84  def ShuffleTheDeck():
85      for i in range(len(mydeck._cards)-1, 0, -1):
86
87          # Pick a random index from 0 to i
88          j = random.randint(0, i + 1)
89
90          # Swap arr[i] with the element at random index
91          mydeck._cards[i], mydeck._cards[j] = mydeck._cards[j], mydeck._cards[i]
```

This code is the Fisher-Yates shuffle algorithm takes the length of range of the deck of cards (52), writes down every value as a new list (i), and works its way down by incrementing the size of the list down in increments of -1.

J is a random integer between 0 and 1 more than the current length of the deck of cards. The values picked are what gets removed from the (i) list and appended to the (j) list

Once the lists have been worked through, the (j) list overwrites the (I) list, replacing the once ordered deck with a new, randomised version.

4: Time to test it!
We will use the ShowTheDeck() function to show the deck - then, we will shuffle, and then show the deck again

```
ShowTheDeck()
ShuffleTheDeck()
ShowTheDeck()
```

PART 3
MOVING CARDS FROM THE DECK TO THE HAND

1:
Create a hand object = we will give the hand an empty list to move cards into, and its own value, which will be the total value of the cards moved into the hand. We will do this using the card values we created earlier

Note that we are using self to refer to the hands own values again

```
99  class Hand:
100     def __init__(self, handvalue): #tell it to have self referential hand value
101         self.cards = []
102         self.handvalue = handvalue
```

2:
Then, we will create an instance of the hand - remember we need to add the initial value of the hand when it is initialised - this is always 0 because there are no cards in the hand yet!

```
106 myhand = Hand(0)
```

3: Lets deal a card = we will do this using a method.
We create a taken card variable to move a card into temporarily - this will allow us to pull the cards value and add it to the hand before moving the card itself into the hand.
We will first pop the card from the deck, using position 0 - this is always the first position in the list, and so is the first position in the deck. Lastly, the card we popped is now appended to the hand.

```
def MyDeal(): #create a method to deal the card
    takencard = mydeck._cards.pop (0) #pop the card from the list so we can hold it in memory
    to pull the value
    myhand.handvalue += takencard._value # increase the value of the hand by the value of the
    card
    myhand.cards.append(takencard)
```

3:
Now we create a method to show what is in the hand
We will use f strings to show what cards are in the hand, then how much the hand is worth by calling the variables we created for the hand objects earlier.

```
def ShowMyHand():
    print (f"I have {myhand.cards} in my hand")
    print (f"my hand is now worth {myhand.handvalue}") #show hand value
```

4:
Let's test it!
Using the methods we have created so far, let's create a for loop that deals a card and shows our hand twice over

```
for i in (1,2):
    MyDeal() # deal a card
    ShowMyHand()
```

PART 4: MORE HANDS

There's usually more than one payer in a card game - so lets find out how to create more hands, and send cards to those hands too!

1:
First, let's create another instance of a hand object

```
otherhand = Hand(0)
```

2:
Create another function so that cards can be dealt to a different player:

```
def OtherDeal():
    takencard = mydeck._cards.pop (0) #pop the card from the list so we can hold it in memory
    to pull the value
    otherhand.handvalue += takencard._value # increase the value of the hand by the value of
    the card
    otherhand.cards.append(takencard)
```

Notice that this time, we refer to the otherhand - remember that because the other hand is an instance of a hand, it has its own value and list of held cards.

3:
Create another method that shows you the value of the other hand and what cards are in it

```
def ShowOtherHand():
    print (f"The cards in the  other hand are {otherhand.cards}")
    print (f"The value of the other hand is {otherhand.handvalue} \n")
```

In a game you may not want to see this information, but for now we have it to show how the process works

4:

Let's test it!

This time, lets make two for loops - first to deal to you, then to the other hand. Then we can see the values of your hand and the other hand

```
ShuffleTheDeck() #always shuffle the deck before dealing!
for i in (1,2):
    MyDeal()
ShowMyHand()
for i in (1,2):
    OtherDeal()
ShowOtherHand()
```