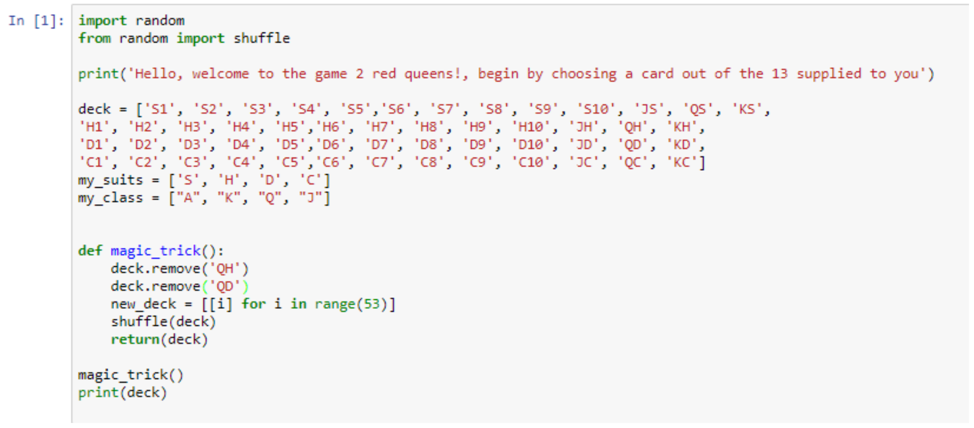
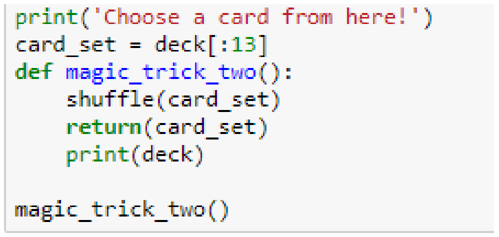
The task which was given to us was to re-create a magic trick in which the user chooses a card and it is found within the two red queens of the card deck. This fits in with computational thinking as we have to think of various ways to recreate human actions such as shuffling cards and placing cards in at certain points.



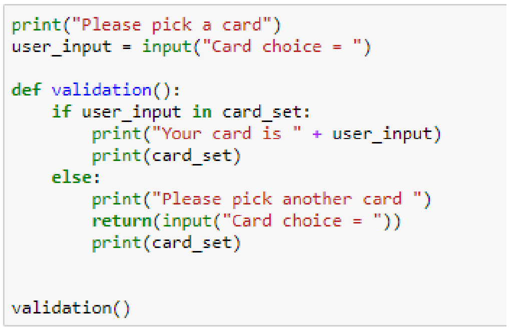
```
In [1]: import random
        from random import shuffle

        print('Hello, welcome to the game 2 red queens!, begin by choosing a card out of the 13 supplied to you')

        deck = ['S1', 'S2', 'S3', 'S4', 'S5','S6', 'S7', 'S8', 'S9', 'S10', 'JS', 'QS', 'KS',
        'H1', 'H2', 'H3', 'H4', 'H5','H6', 'H7', 'H8', 'H9', 'H10', 'JH', 'QH', 'KH',
        'D1', 'D2', 'D3', 'D4', 'D5','D6', 'D7', 'D8', 'D9', 'D10', 'JD', 'QD', 'KD',
        'C1', 'C2', 'C3', 'C4', 'C5','C6', 'C7', 'C8', 'C9', 'C10', 'JC', 'QC', 'KC']
        my_suits = ['S', 'H', 'D', 'C']
        my_class = ["A", "K", "Q", "J"]


        def magic_trick():
            deck.remove('QH')
            deck.remove('QD')
            new_deck = [[i] for i in range(53)]
            shuffle(deck)
            return(deck)

        magic_trick()
        print(deck)
```

I started by importing the two python libraries 'random' and 'shuffle' as random would be useful in selecting random cards and shuffle would be useful for when card decks need to be shuffled. I then went on to create the deck using a list along with the suits and classes as the decks would be a very important part of this operation. We then see a function in which I take out the two red queens to make sure that they are not shuffled along with the other cards. This is done with the deck.remove(). To shuffle the cards, I decided to use the range function which in my code states 'for cards up to position 53', they are then shuffled using shuffle(deck) and the function is called and the deck is then printed.

This function allows 13 random cards to be selected from the previously shuffled deck. These cards are then shuffled themselves and printed when the function is called.



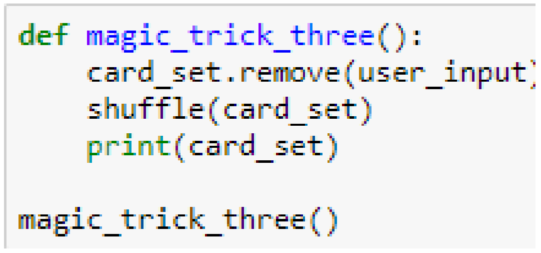
```
print('Choose a card from here!')
card_set = deck[:13]
def magic_trick_two():
    shuffle(card_set)
    return(card_set)
    print(deck)

magic_trick_two()
```



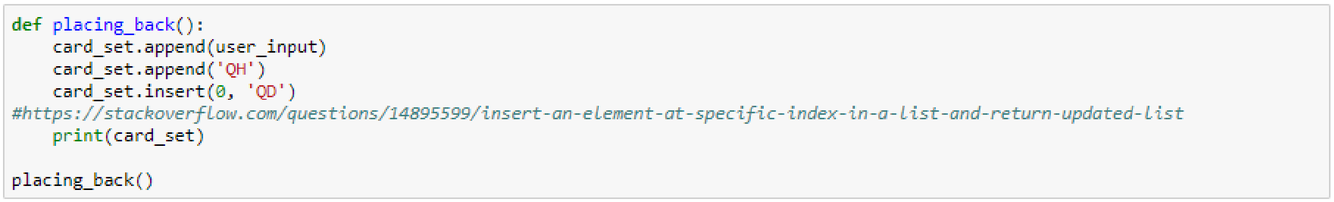
```
print("Please pick a card")
user_input = input("Card choice = ")

def validation():
    if user_input in card_set:
        print("Your card is " + user_input)
        print(card_set)
    else:
        print("Please pick another card ")
        return(input("Card choice = "))
        print(card_set)


validation()
```

Here we validate the users card choice via their input. The function checks as to whether if the user input is in the card set printing 'Your card is' + the user input through the if statement if the card is correct or asks the user to pick another card with the else statement if the card chosen isn't within the card set

With this function we ca shuffle the cards without the users card as we need to place the users card at a certain position later on in the trick. We then have a shuffled deck of our currently 12 cards.

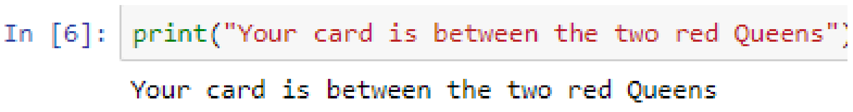


```
def magic_trick_three():
    card_set.remove(user_input
    shuffle(card_set)
    print(card_set)

magic_trick_three()
```



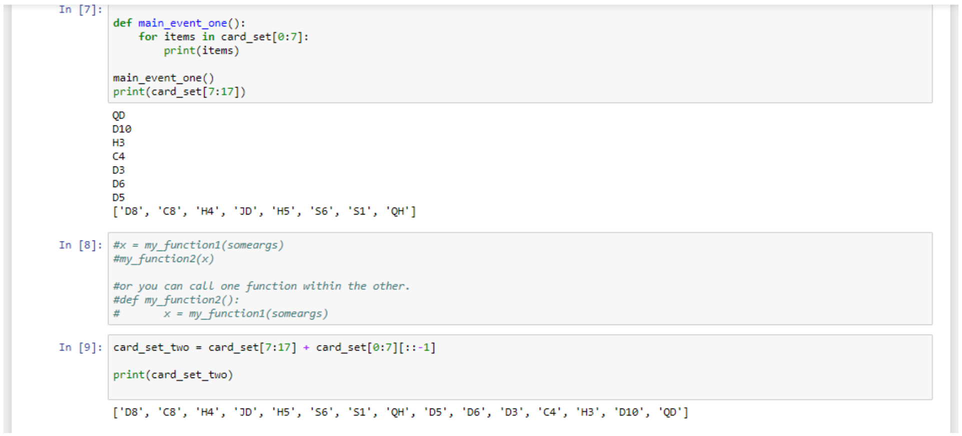
```
def placing_back():
    card_set.append(user_input)
    card_set.append('QH')
    card_set.insert(0, 'QD')
#https://stackoverflow.com/questions/14895599/insert-an-element-at-specific-index-in-a-list-and-return-updated-list
    print(card_set)

placing_back()
```

Now I had to place back the users chosen card along with the two red queens. I placed the users card down first as I knew it would be placed at the end and then to place the 'QH' card behind it simply just placed it after in the function guaranteeing its place at the end. To place the other red queen 'QD' at the front I had to look to stack overflow for my solution which stated how if I used *card_set.insert(0, 'QD')* with the zero stating how I wanted it at the front and insert allowing for me to place 'QD' at that certain position.



```
In [6]: print("Your card is between the two red Queens")

        Your card is between the two red Queens
```

In this statement we tell the user that their card is between the two red queens and proceed to shuffle the cards.

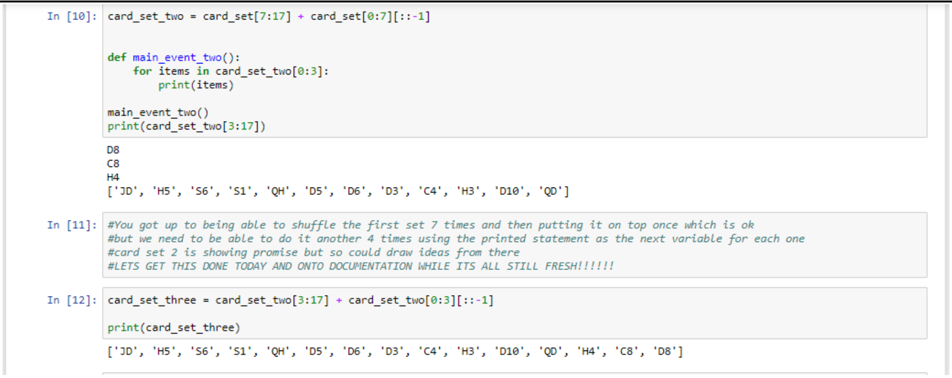


```
In [7]: def main_event_one():
            for items in card_set[0:7]:
                print(items)

        main_event_one()
        print(card_set[7:17])

        QD
        D10
        H3
        C4
        D3
        D6
        D5
        ['D8', 'C8', 'H4', 'JD', 'H5', 'S6', 'S1', 'QH']

In [8]: #x = my_function1(someargs)
        #my_function2(x)

        #or you can call one function within the other.
        #def my_function2():
        #       x = my_function1(someargs)

In [9]: card_set_two = card_set[7:17] + card_set[0:7][::-1]

        print(card_set_two)

        ['D8', 'C8', 'H4', 'JD', 'H5', 'S6', 'S1', 'QH', 'D5', 'D6', 'D3', 'C4', 'H3', 'D10', 'QD']
```

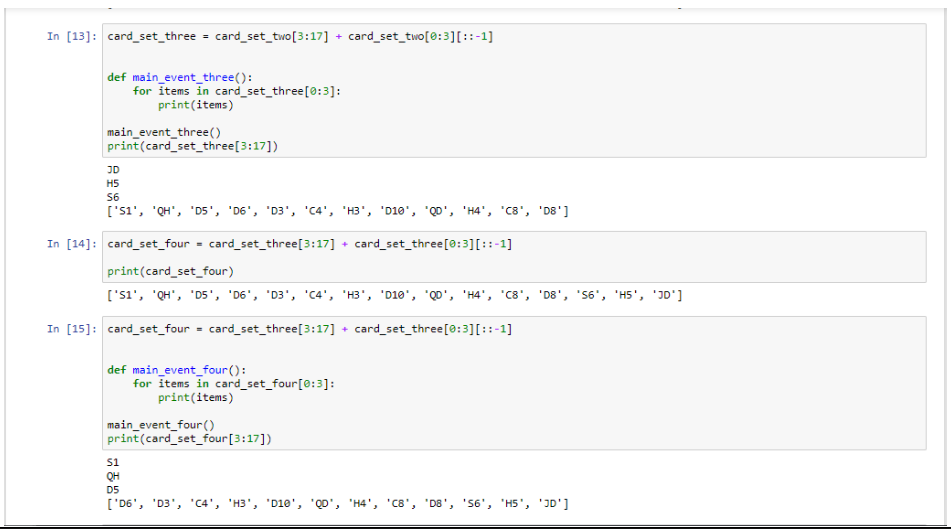
Commented out section is a draft of ideas as I struggled to figure out a solution for this section.

Here we begin the main part of the trick where we have to place down each of the cards one by one and then place the leftover stack of cards on the top. I did this by taking the first 7 items of the list and printed them out separate to those leftover in the function. I then had to repeat this action with the result from this function so I had to create a new card set and use [::-1] which is another Idea I saw on stack overflow. This reverses the list order however allowed me to put the cards on in the correct order.

Commented out section was a quick note to myself to remind myself where I had got up to for when I returned to the project.

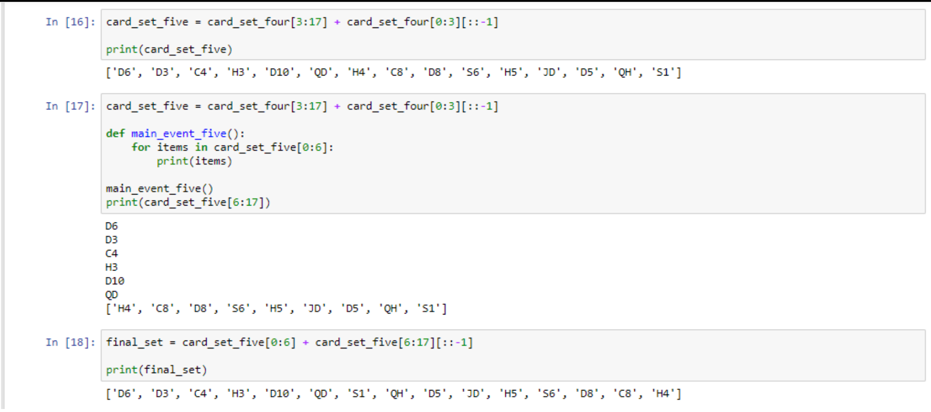


```
In [10]: card_set_two = card_set[7:17] + card_set[0:7][::-1]


         def main_event_two():
             for items in card_set_two[0:3]:
                 print(items)

         main_event_two()
         print(card_set_two[3:17])

         D8
         C8
         H4
         ['JD', 'H5', 'S6', 'S1', 'QH', 'D5', 'D6', 'D3', 'C4', 'H3', 'D10', 'QD']

In [11]: #You got up to being able to shuffle the first set 7 times and then putting it on top once which is ok
         #but we need to be able to do it another 4 times using the printed statement as the next variable for each one
         #card set 2 is showing promise but so could draw ideas from there
         #LETS GET THIS DONE TODAY AND ONTO DOCUMENTATION WHILE ITS ALL STILL FRESH!!!!!!

In [12]: card_set_three = card_set_two[3:17] + card_set_two[0:3][::-1]

         print(card_set_three)

         ['JD', 'H5', 'S6', 'S1', 'QH', 'D5', 'D6', 'D3', 'C4', 'H3', 'D10', 'QD', 'H4', 'C8', 'D8']
```



```
In [13]: card_set_three = card_set_two[3:17] + card_set_two[0:3][::-1]


         def main_event_three():
             for items in card_set_three[0:3]:
                 print(items)

         main_event_three()
         print(card_set_three[3:17])

         JD
         H5
         S6
         ['S1', 'QH', 'D5', 'D6', 'D3', 'C4', 'H3', 'D10', 'QD', 'H4', 'C8', 'D8']

In [14]: card_set_four = card_set_three[3:17] + card_set_three[0:3][::-1]

         print(card_set_four)

         ['S1', 'QH', 'D5', 'D6', 'D3', 'C4', 'H3', 'D10', 'QD', 'H4', 'C8', 'D8', 'S6', 'H5', 'JD']

In [15]: card_set_four = card_set_three[3:17] + card_set_three[0:3][::-1]


         def main_event_four():
             for items in card_set_four[0:3]:
                 print(items)

         main_event_four()
         print(card_set_four[3:17])

         S1
         QH
         D5
         ['D6', 'D3', 'C4', 'H3', 'D10', 'QD', 'H4', 'C8', 'D8', 'S6', 'H5', 'JD']
```

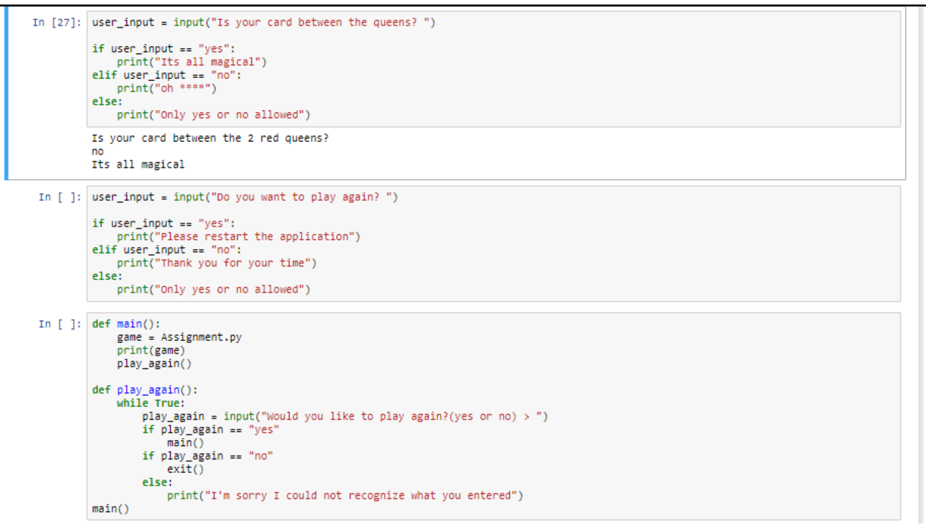
Again you can see how I used the same function each time however just used different numbers for when the dealer would be finished counting that specific amount of cards. Making new variables and using the [::-1] operator also allowed me to put the cards which I counted at the back in the correct order which was crucial to the tricks success



```
In [16]: card_set_five = card_set_four[3:17] + card_set_four[0:3][::-1]

         print(card_set_five)

         ['D6', 'D3', 'C4', 'H3', 'D10', 'QD', 'H4', 'C8', 'D8', 'S6', 'H5', 'JD', 'D5', 'QH', 'S1']

In [17]: card_set_five = card_set_four[3:17] + card_set_four[0:3][::-1]

         def main_event_five():
             for items in card_set_five[0:6]:
                 print(items)

         main_event_five()
         print(card_set_five[6:17])

         D6
         D3
         C4
         H3
         D10
         QD
         ['H4', 'C8', 'D8', 'S6', 'H5', 'JD', 'D5', 'QH', 'S1']

In [18]: final_set = card_set_five[0:6] + card_set_five[6:17][::-1]

         print(final_set)

         ['D6', 'D3', 'C4', 'H3', 'D10', 'QD', 'S1', 'QH', 'D5', 'JD', 'H5', 'S6', 'D8', 'C8', 'H4']
```

Again you are able to see what I have previously stated above

Finally, we have a user input which asks the user whether their card is within the two red queens. This has a range of responses which are humorous but also an else statement to make sure and remind the user to enter a valid response.



```
In [27]: user_input = input("Is your card between the queens? ")

         if user_input == "yes":
             print("Its all magical")
         elif user_input == "no":
             print("oh ****")
         else:
             print("Only yes or no allowed")

         Is your card between the 2 red queens?
         no
         Its all magical

In [ ]: user_input = input("Do you want to play again? ")

        if user_input == "yes":
            print("Please restart the application")
        elif user_input == "no":
            print("Thank you for your time")
        else:
            print("Only yes or no allowed")

In [ ]: def main():
            game = Assignment.py
            print(game)
            play_again()

        def play_again():
            while True:
                play_again = input("Would you like to play again?(yes or no) > ")
                if play_again == "yes"
                    main()
                if play_again == "no"
                    exit()
                else:
                    print("I'm sorry I could not recognize what you entered")
        main()
```

We then have an if statement and a function of which I opted for the function. This function takes the game as a variable and simply says that if the user says yes when asked whether they want to play again then the game should restart and the card deck should show up as we see at the start of the game, however if they decide not to then the program should exit and close. This is a solution I acquired from stack overflow and have referenced fully in my ATOM code.