Assessment Module	RGU CM 1601 Programming Fundamentals
Module Leader	Rajitha Jayasinghe
Student Name	Lasal Chathranjey Jayawardena
RGU Id	2016279
IIT Id	20200344
Submission Type	Coursework Part 1
Submission Date	10th December 2020
IDE	Visual Studio Code

Course Work Report

Summary

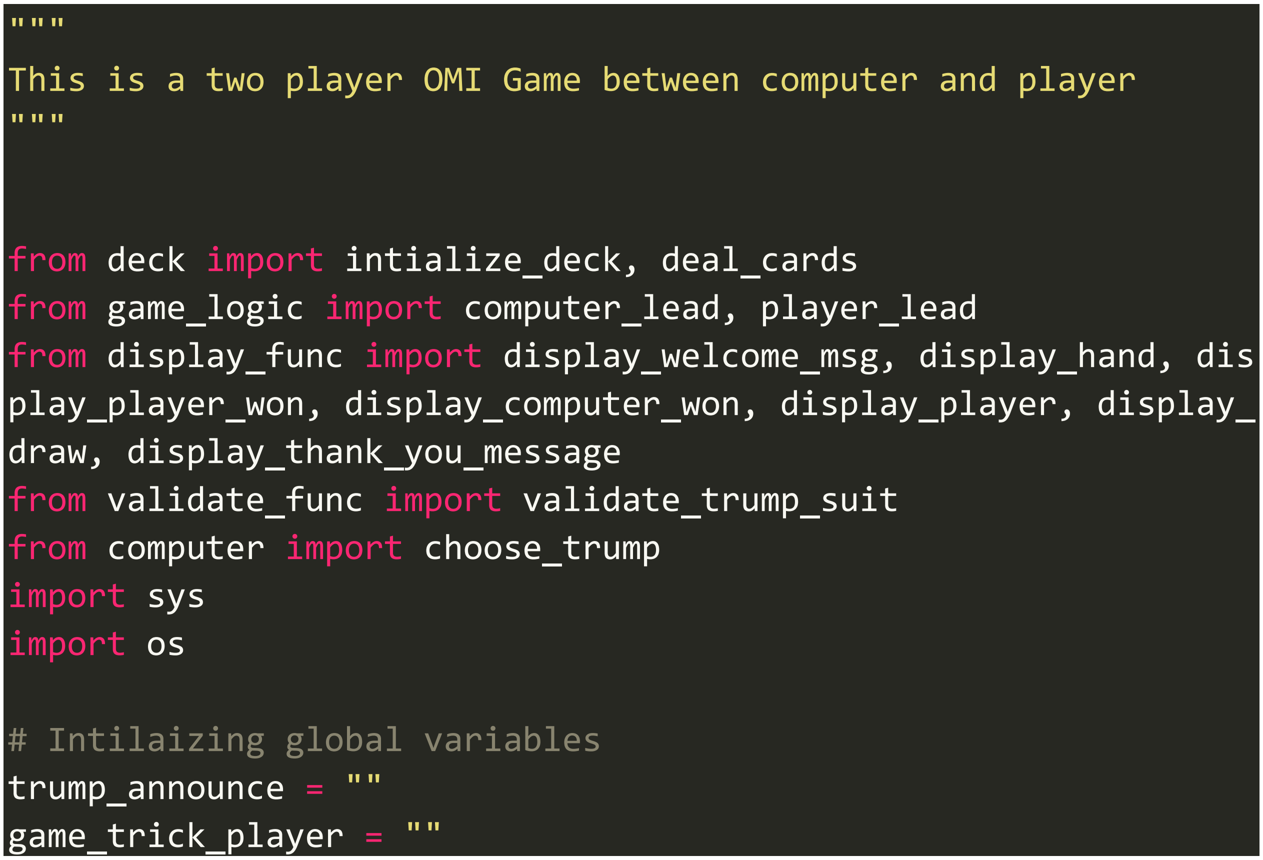
The program written for the coursework is an interactive command line OMI game(modified). In the program, the computer plays the role of the second player and also the game administrator as well. There is a separate folder for the game source code, 1 pdf document on the flowchart of the shuffling algorithm and finally another pdf document on the flowchart of the game phase (8 tricks after the game deck is shuffled).

Improvements

1. Additional functionality where each trick player is tracked even after the game is closed and then opened (using text files).

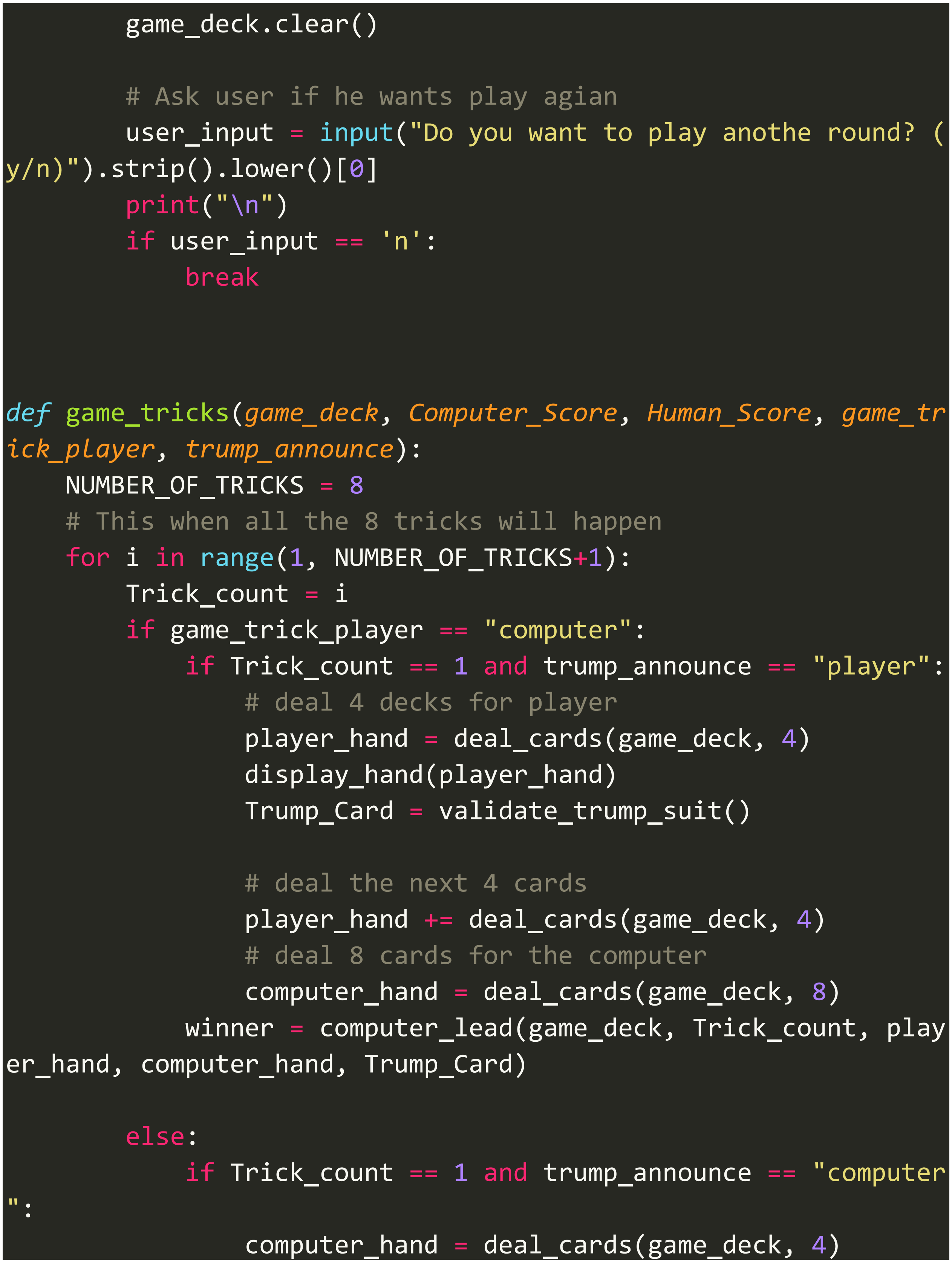
2. Added ASCII art for Welcome message, Result message, thank you message.
3. Changed how the user decks are printed into a more user friendly readable format.
4. Implemented the test cases using try, except and assertion with test cases for function which checks the trick winner.
5. Incorporation of meaningful messages to user when the game rules are not followed.
6. Completely implemented functional decomposition and modularization.
7. Further Strengthened the logic when computer chooses cards (e.g. If there are higher cards than the users' card then selects the lower out of all).
8. All functions and algorithms written with balanced consideration of Big-Oh, code-readability and memory used.
9. Doc strings used to describe each game module made and meaningful comments used for assistance in understanding code.
10. Added an extra sanitation function for user input for flexibility and improved user experience.

app.py code – This the main part of the program where the entire game happens.

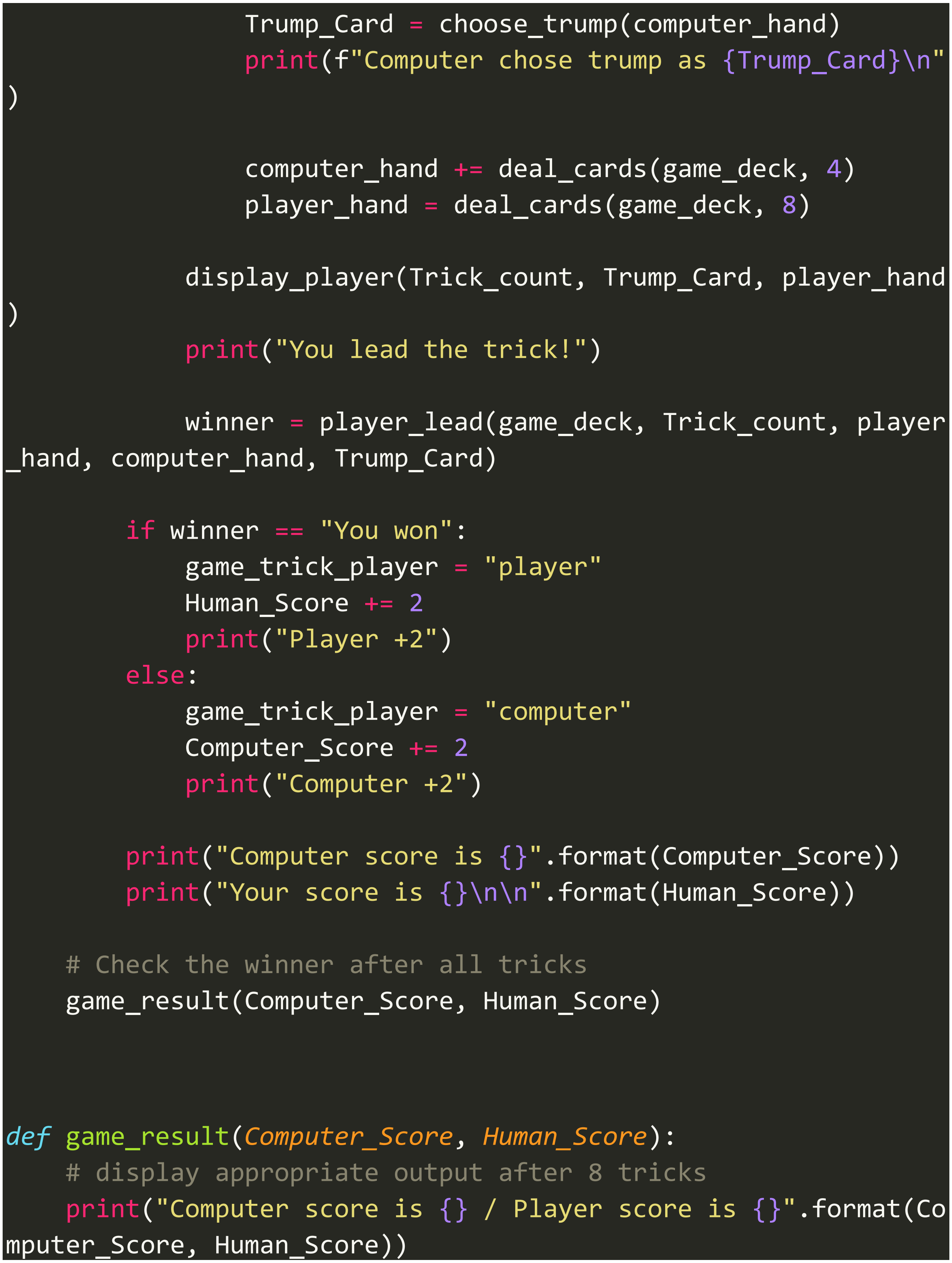


```python
"""
This is a two player OMI Game between computer and player
"""


from deck import intialize_deck, deal_cards
from game_logic import computer_lead, player_lead
from display_func import display_welcome_msg, display_hand, display_player_won, display_computer_won, display_player, display_draw, display_thank_you_message
from validate_func import validate_trump_suit
from computer import choose_trump
import sys
import os

# Intilaizing global variables
trump_announce = ""
game_trick_player = ""
```

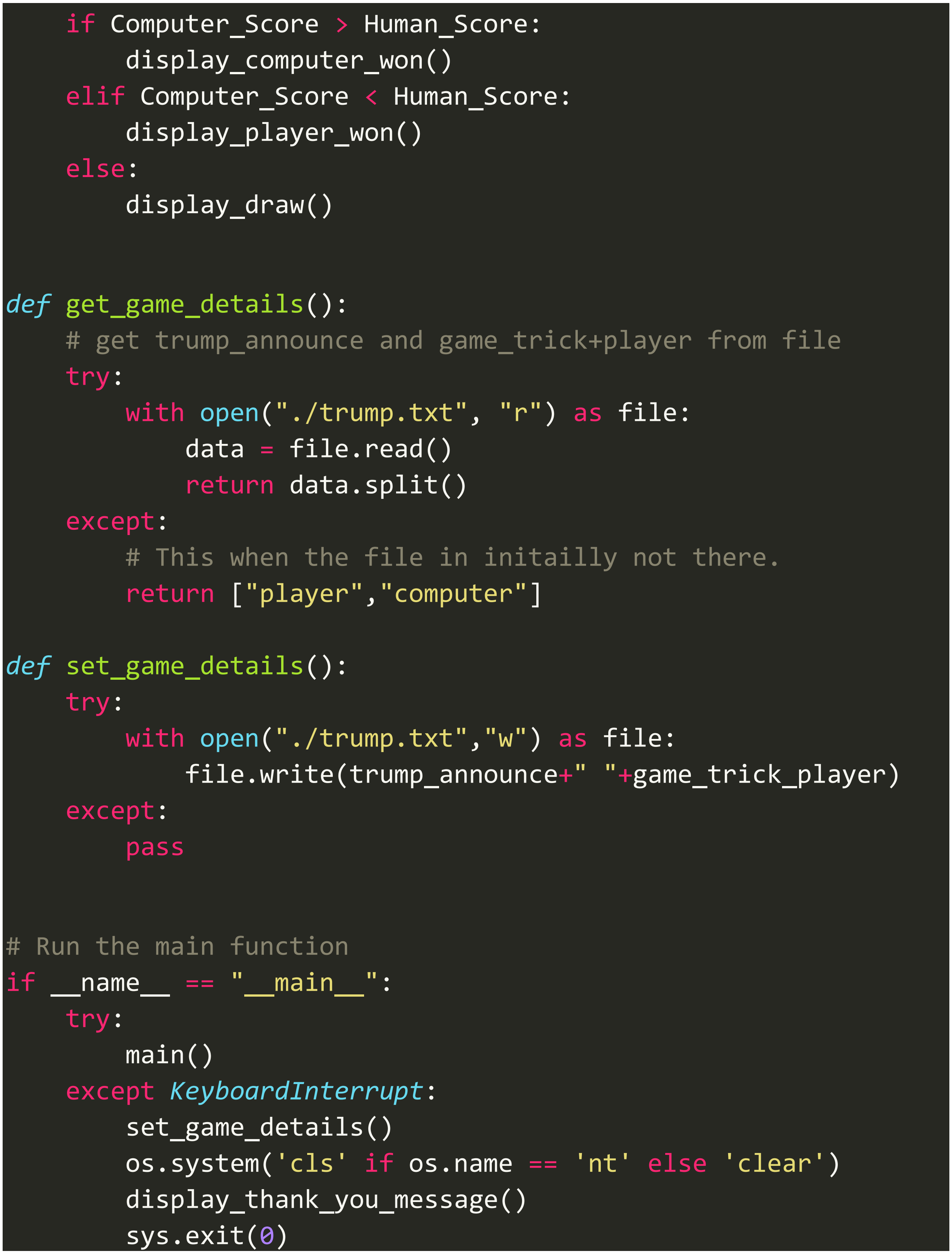
```
# End of Global Variables

def main():

    # Displays Welcome message
    display_welcome_msg()
    # This is to identify who will tell the trumps
    global trump_announce
    # This is to identify who will lead the trick, this changes every trick
    global game_trick_player

    trump_announce, game_trick_player = get_game_details()

    while True:
        # intialize scores and the game deck
        Computer_Score = 0
        Human_Score = 0
        game_deck = intialize_deck()

        # This when all 8 tricks begin, and the most of the game logic is implemented
        game_tricks(game_deck, Computer_Score, Human_Score,game_trick_player, trump_announce)

        # Switch trump annouonce and game_trick players after 8 tricks, when player wants to play agian
        if trump_announce == "player":
            trump_announce = "computer"
            game_trick_player = "player"
        else:
            trump_announce = "player"
            game_trick_player = "computer"

        # Clear deck after each round( after 8 tricks )
```



```
        game_deck.clear()

        # Ask user if he wants play agian
        user_input = input("Do you want to play anothe round? (
y/n)").strip().lower()[0]
        print("\n")
        if user_input == 'n':
            break



def game_tricks(game_deck, Computer_Score, Human_Score, game_tr
ick_player, trump_announce):
    NUMBER_OF_TRICKS = 8
    # This when all the 8 tricks will happen
    for i in range(1, NUMBER_OF_TRICKS+1):
        Trick_count = i
        if game_trick_player == "computer":
            if Trick_count == 1 and trump_announce == "player":
                # deal 4 decks for player
                player_hand = deal_cards(game_deck, 4)
                display_hand(player_hand)
                Trump_Card = validate_trump_suit()

                # deal the next 4 cards
                player_hand += deal_cards(game_deck, 4)
                # deal 8 cards for the computer
                computer_hand = deal_cards(game_deck, 8)
            winner = computer_lead(game_deck, Trick_count, play
er_hand, computer_hand, Trump_Card)

        else:
            if Trick_count == 1 and trump_announce == "computer
":

                computer_hand = deal_cards(game_deck, 4)
```



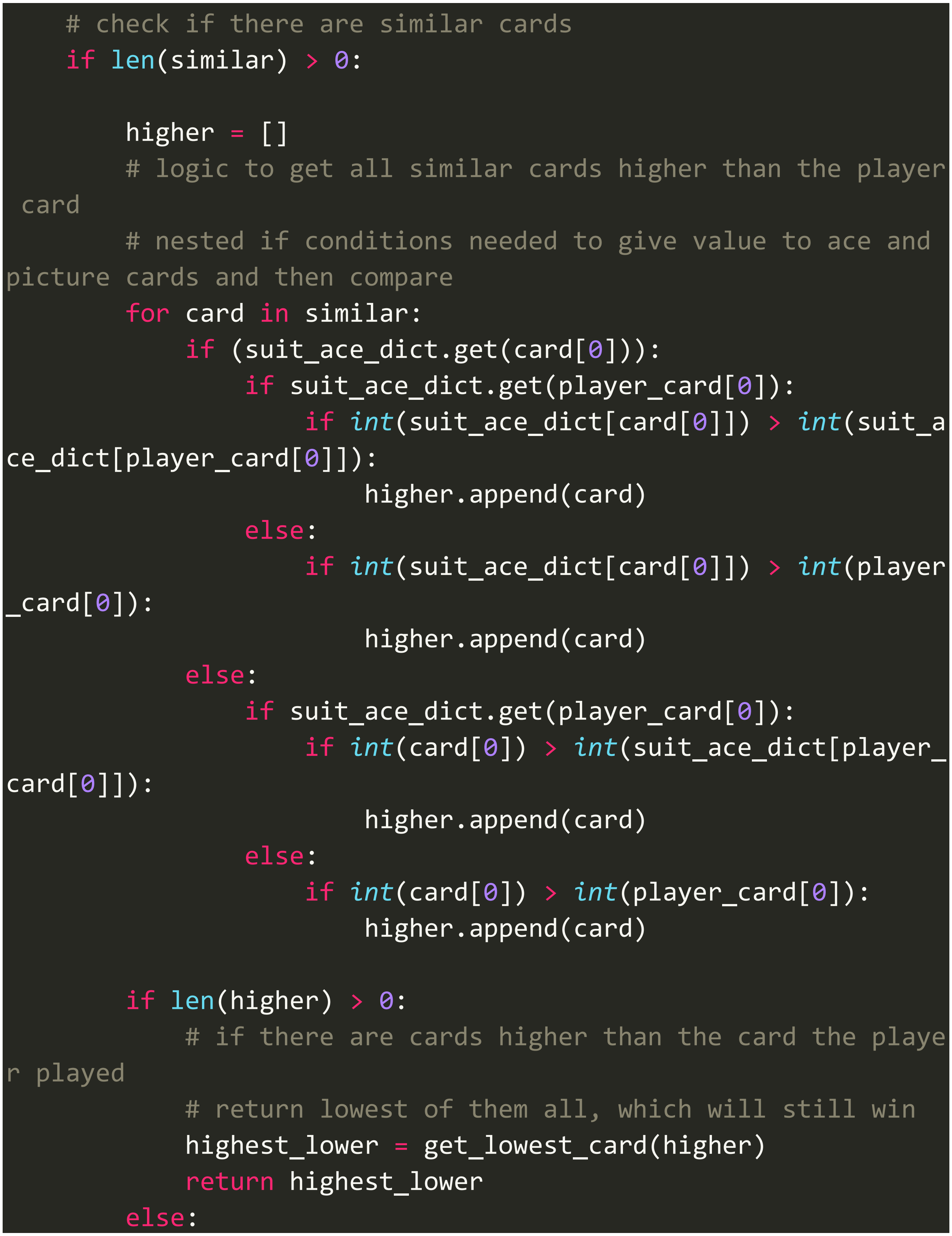
```
                Trump_Card = choose_trump(computer_hand)
                print(f"Computer chose trump as {Trump_Card}\n"
)

                computer_hand += deal_cards(game_deck, 4)
                player_hand = deal_cards(game_deck, 8)

            display_player(Trick_count, Trump_Card, player_hand
)
            print("You lead the trick!")

            winner = player_lead(game_deck, Trick_count, player
_hand, computer_hand, Trump_Card)

        if winner == "You won":
            game_trick_player = "player"
            Human_Score += 2
            print("Player +2")
        else:
            game_trick_player = "computer"
            Computer_Score += 2
            print("Computer +2")

        print("Computer score is {}".format(Computer_Score))
        print("Your score is {}\n\n".format(Human_Score))

    # Check the winner after all tricks
    game_result(Computer_Score, Human_Score)



def game_result(Computer_Score, Human_Score):
    # display appropriate output after 8 tricks
    print("Computer score is {} / Player score is {}".format(Co
mputer_Score, Human_Score))
```



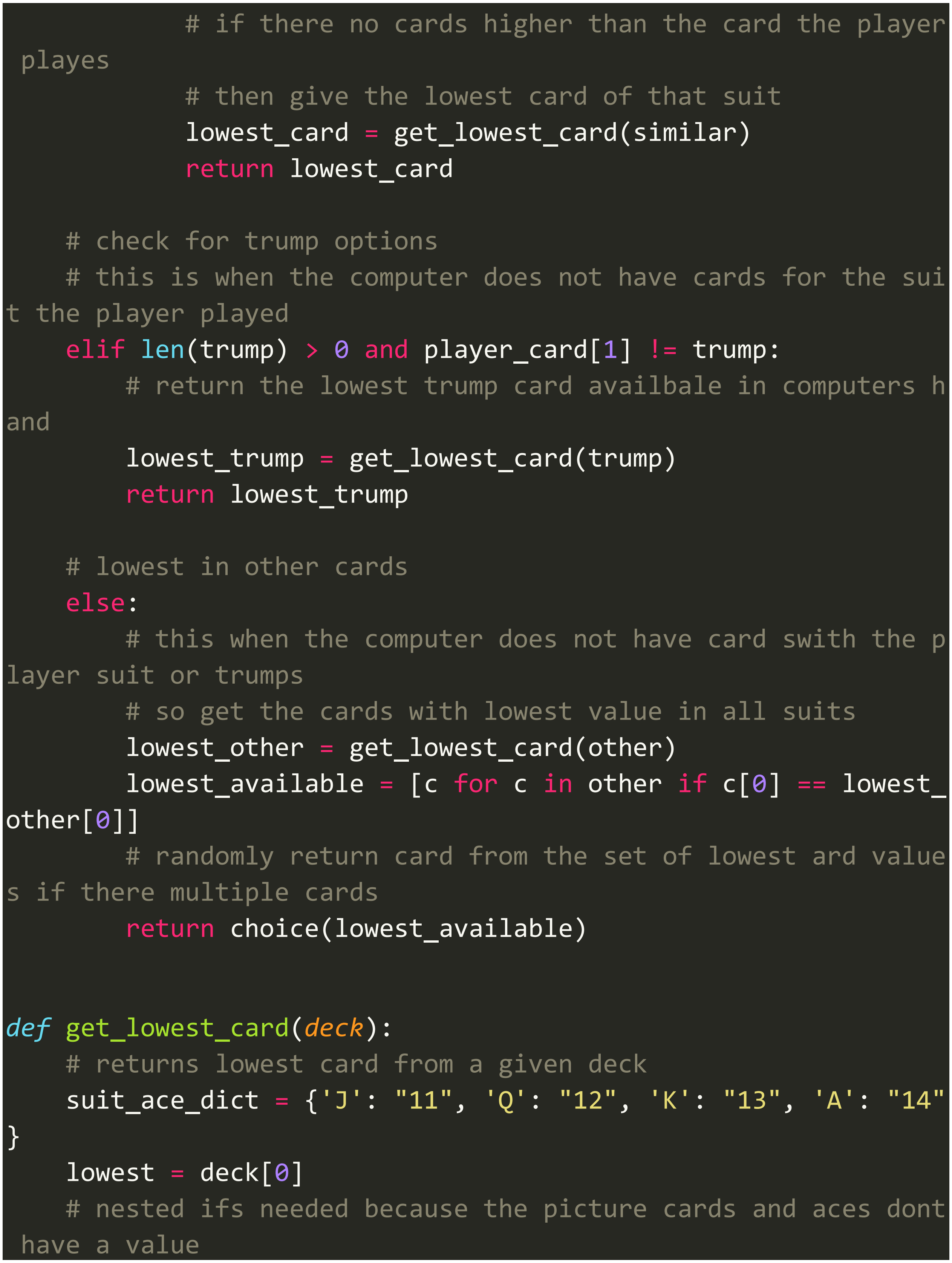
```python
    if Computer_Score > Human_Score:
        display_computer_won()
    elif Computer_Score < Human_Score:
        display_player_won()
    else:
        display_draw()


def get_game_details():
    # get trump_announce and game_trick+player from file
    try:
        with open("./trump.txt", "r") as file:
            data = file.read()
            return data.split()
    except:
        # This when the file in initailly not there.
        return ["player","computer"]

def set_game_details():
    try:
        with open("./trump.txt","w") as file:
            file.write(trump_announce+" "+game_trick_player)
    except:
        pass


# Run the main function
if __name__ == "__main__":
    try:
        main()
    except KeyboardInterrupt:
        set_game_details()
        os.system('cls' if os.name == 'nt' else 'clear')
        display_thank_you_message()
        sys.exit(0)
    else:
        set_game_details()
        os.system('cls' if os.name == 'nt' else 'clear')
        display_thank_you_message()
```

computer.py code - This is where the functions done by the computer as a player are stored.

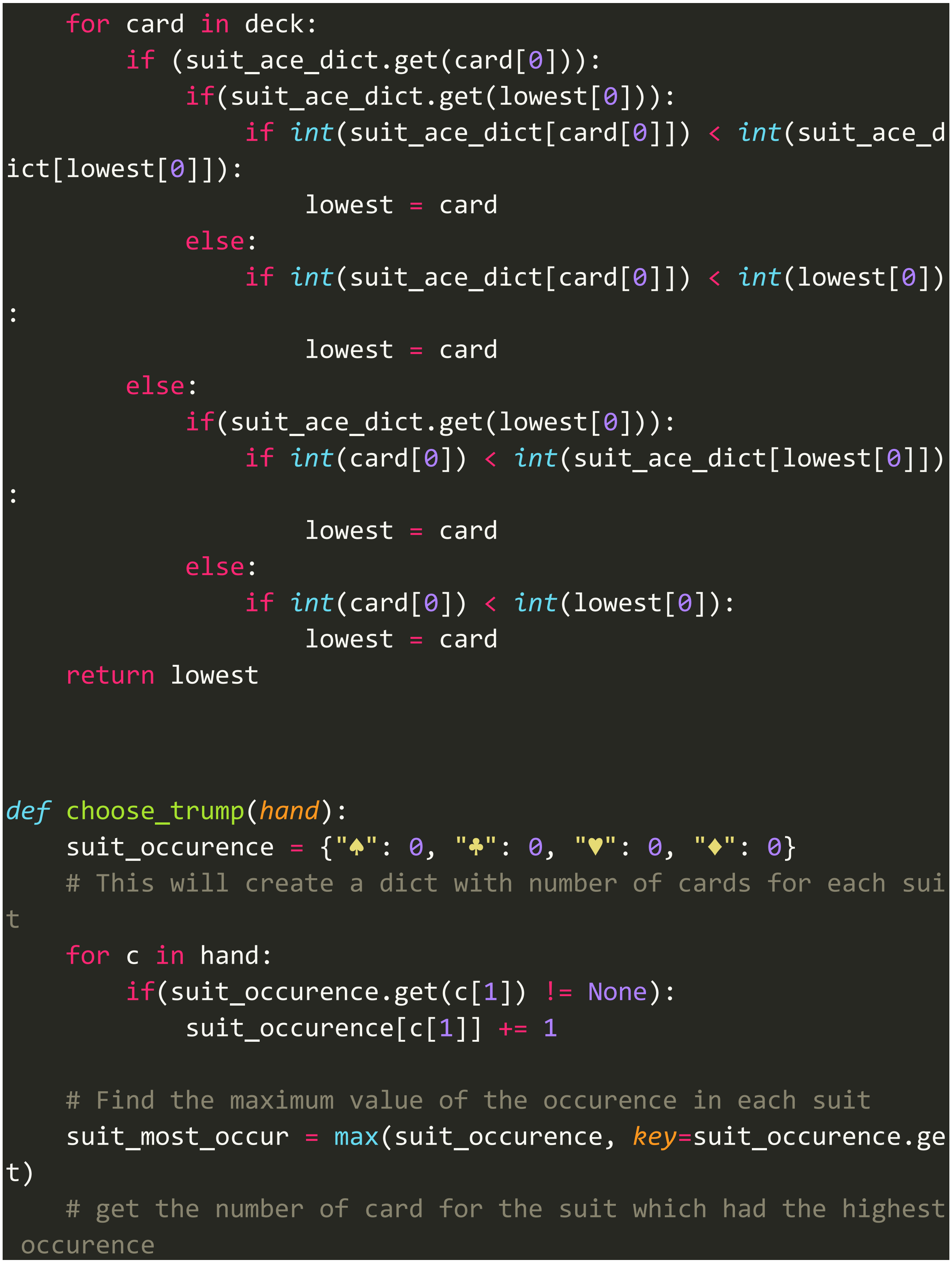
```
"""
This is where most of the function needed by the computer is written
"""

from random import choice

def play_card(card_deck):
    # This is when the computer's turn to lead the trick
    return choice(card_deck)


def computer_play_card(c_deck, player_card, trump):
    suit_ace_dict = {'J': "11", 'Q': "12", 'K': "13", 'A': "14"}
    # check whether similar cards
    similar = [c for c in c_deck if c[1] == player_card[1]]
    # all cards with trump
    trump = [c for c in c_deck if c[1] == trump]
    # all other card other than trumps, usefule when no similar cards
    other = [c for c in c_deck if (c[1] != player_card[1]) and (c[1] != trump)]
```



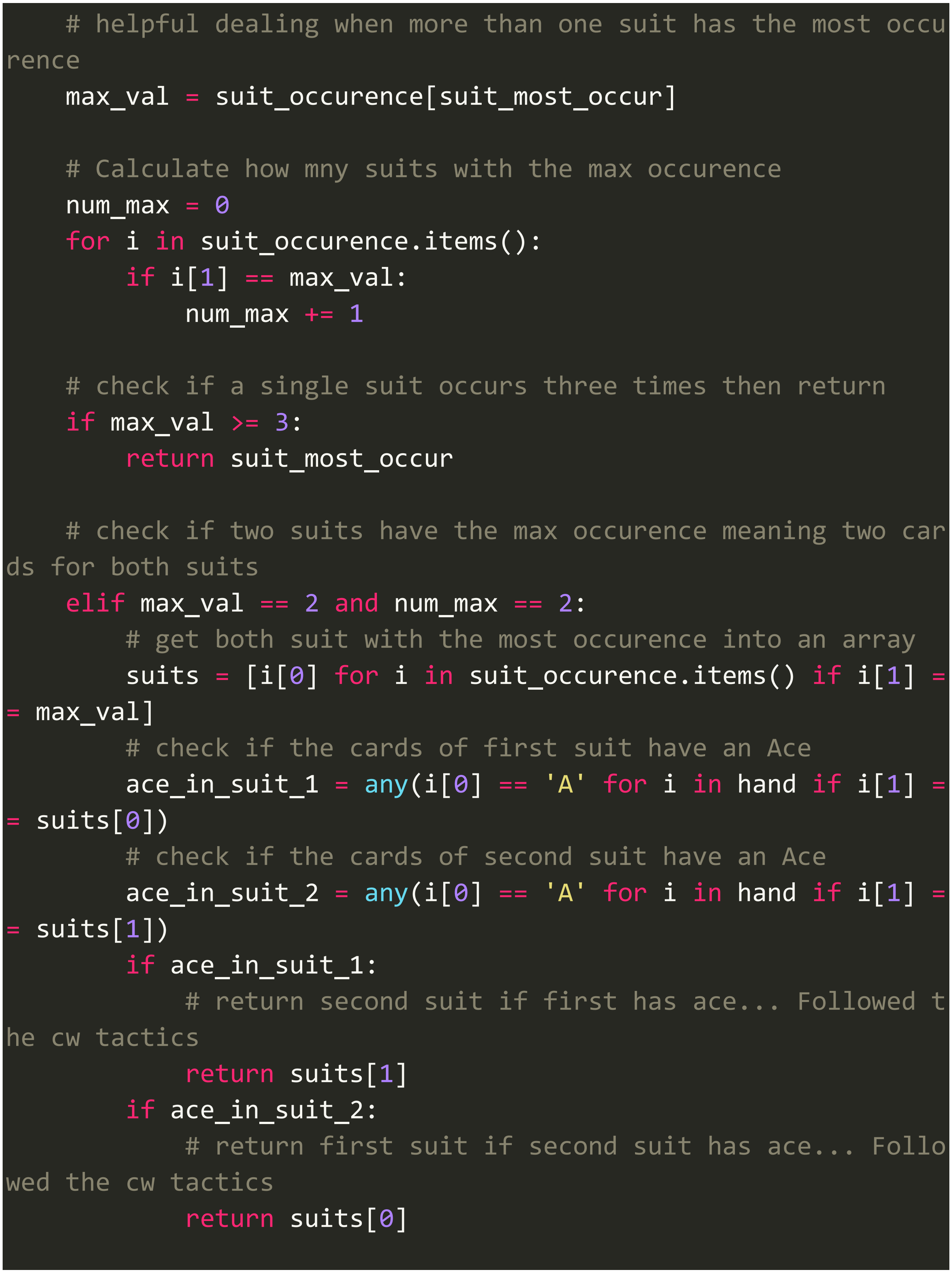
```
    # check if there are similar cards
    if len(similar) > 0:

        higher = []
        # logic to get all similar cards higher than the player
 card
        # nested if conditions needed to give value to ace and
picture cards and then compare
        for card in similar:
            if (suit_ace_dict.get(card[0])):
                if suit_ace_dict.get(player_card[0]):
                    if int(suit_ace_dict[card[0]]) > int(suit_a
ce_dict[player_card[0]]):
                        higher.append(card)
                else:
                    if int(suit_ace_dict[card[0]]) > int(player
_card[0]):
                        higher.append(card)
            else:
                if suit_ace_dict.get(player_card[0]):
                    if int(card[0]) > int(suit_ace_dict[player_
card[0]]):
                        higher.append(card)
                else:
                    if int(card[0]) > int(player_card[0]):
                        higher.append(card)

        if len(higher) > 0:
            # if there are cards higher than the card the playe
r played
            # return lowest of them all, which will still win
            highest_lower = get_lowest_card(higher)
            return highest_lower
        else:
```



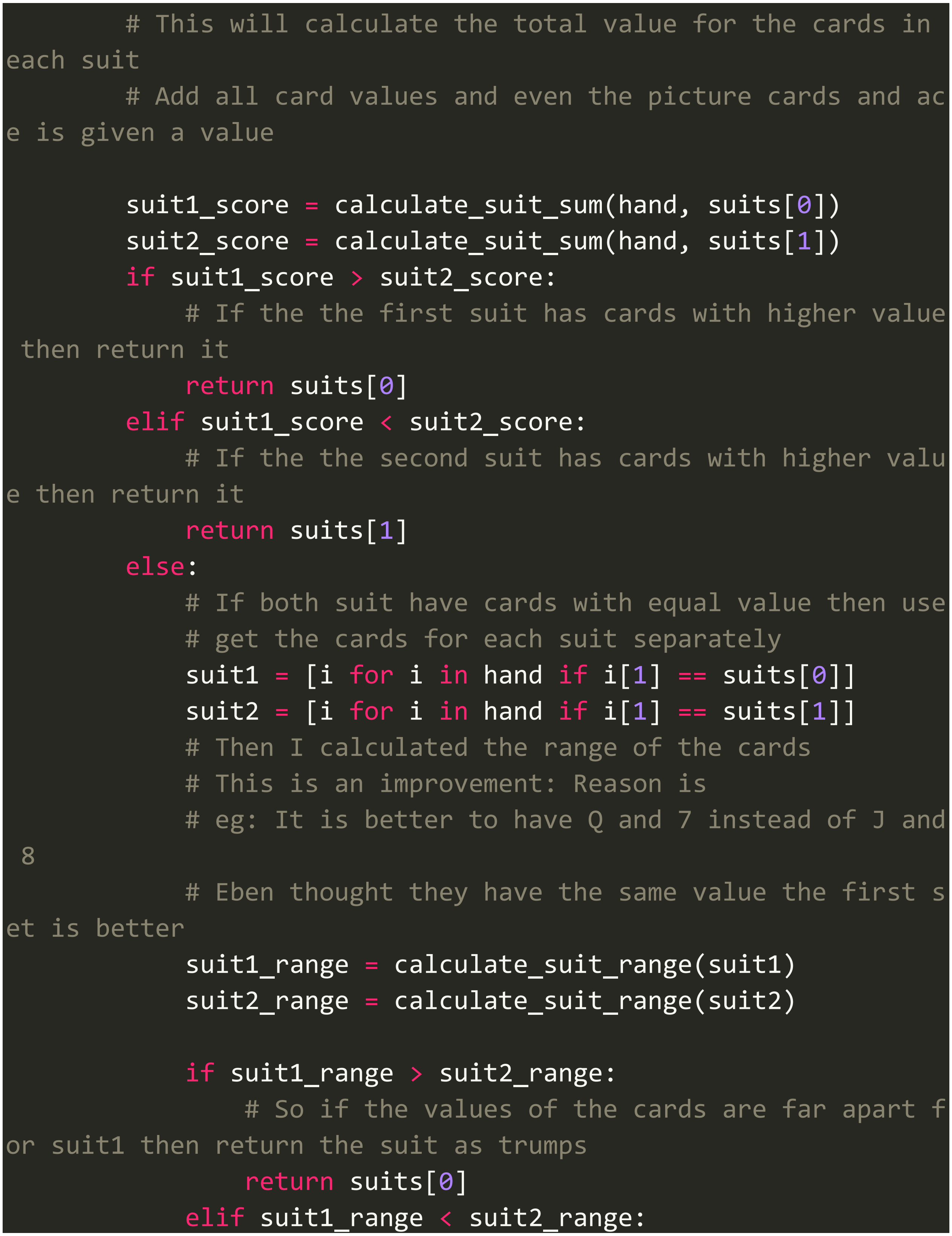
```
            # if there no cards higher than the card the player
 playes
            # then give the lowest card of that suit
            lowest_card = get_lowest_card(similar)
            return lowest_card

    # check for trump options
    # this is when the computer does not have cards for the sui
t the player played
    elif len(trump) > 0 and player_card[1] != trump:
        # return the lowest trump card availbale in computers h
and
        lowest_trump = get_lowest_card(trump)
        return lowest_trump

    # lowest in other cards
    else:
        # this when the computer does not have card swith the p
layer suit or trumps
        # so get the cards with lowest value in all suits
        lowest_other = get_lowest_card(other)
        lowest_available = [c for c in other if c[0] == lowest_
other[0]]
        # randomly return card from the set of lowest ard value
s if there multiple cards
        return choice(lowest_available)


def get_lowest_card(deck):
    # returns lowest card from a given deck
    suit_ace_dict = {'J': "11", 'Q': "12", 'K': "13", 'A': "14"
}
    lowest = deck[0]
    # nested ifs needed because the picture cards and aces dont
 have a value
```



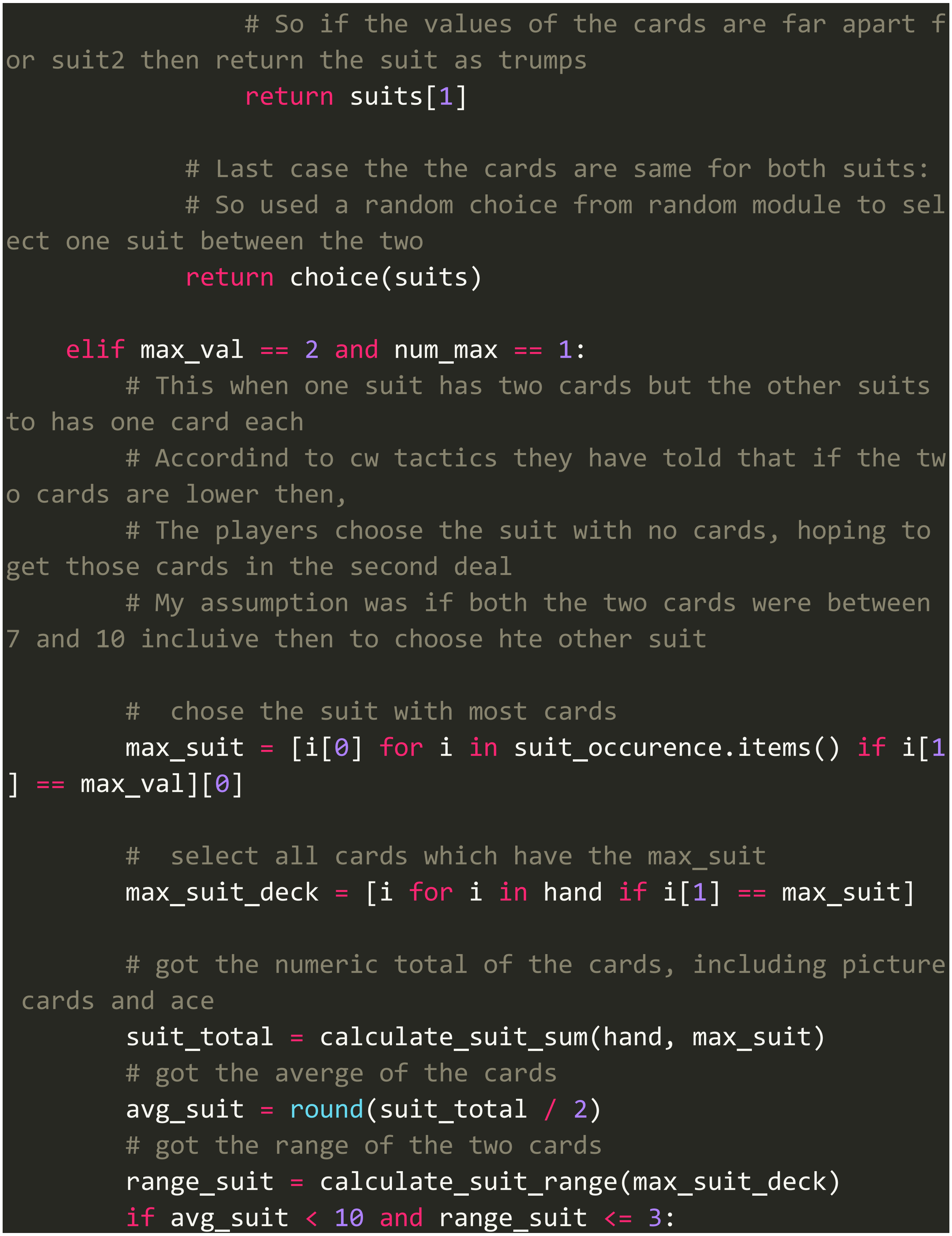
```
    for card in deck:
        if (suit_ace_dict.get(card[0])):
            if(suit_ace_dict.get(lowest[0])):
                if int(suit_ace_dict[card[0]]) < int(suit_ace_d
ict[lowest[0]]):
                    lowest = card
            else:
                if int(suit_ace_dict[card[0]]) < int(lowest[0])
:
                    lowest = card
        else:
            if(suit_ace_dict.get(lowest[0])):
                if int(card[0]) < int(suit_ace_dict[lowest[0]])
:
                    lowest = card
            else:
                if int(card[0]) < int(lowest[0]):
                    lowest = card
    return lowest



def choose_trump(hand):
    suit_occurence = {"♠": 0, "♣": 0, "♥": 0, "♦": 0}
    # This will create a dict with number of cards for each sui
t
    for c in hand:
        if(suit_occurence.get(c[1]) != None):
            suit_occurence[c[1]] += 1

    # Find the maximum value of the occurence in each suit
    suit_most_occur = max(suit_occurence, key=suit_occurence.ge
t)
    # get the number of card for the suit which had the highest
 occurence
```



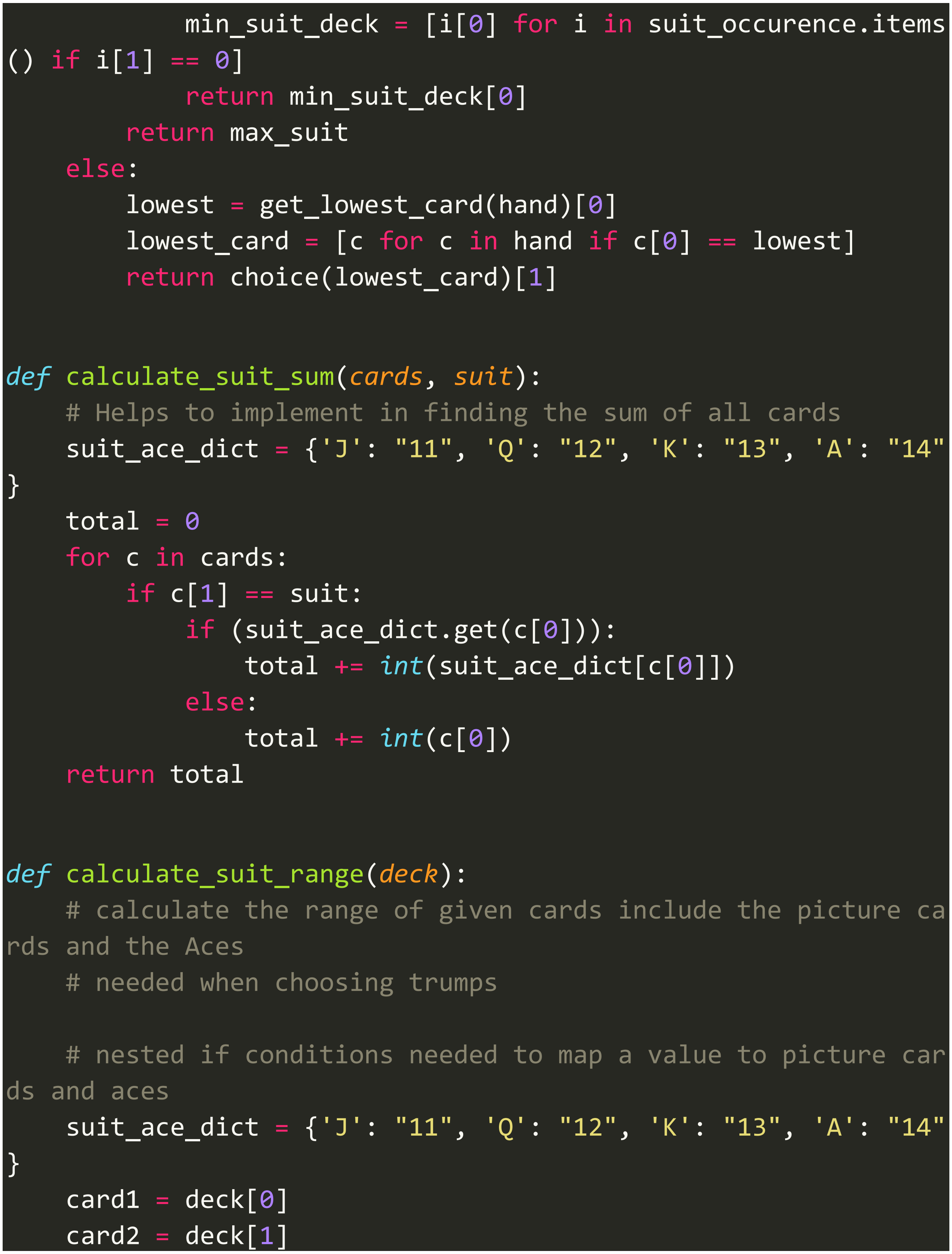
```python
    # helpful dealing when more than one suit has the most occurence
    max_val = suit_occurence[suit_most_occur]

    # Calculate how mny suits with the max occurence
    num_max = 0
    for i in suit_occurence.items():
        if i[1] == max_val:
            num_max += 1

    # check if a single suit occurs three times then return
    if max_val >= 3:
        return suit_most_occur

    # check if two suits have the max occurence meaning two cards for both suits
    elif max_val == 2 and num_max == 2:
        # get both suit with the most occurence into an array
        suits = [i[0] for i in suit_occurence.items() if i[1] == max_val]
        # check if the cards of first suit have an Ace
        ace_in_suit_1 = any(i[0] == 'A' for i in hand if i[1] == suits[0])
        # check if the cards of second suit have an Ace
        ace_in_suit_2 = any(i[0] == 'A' for i in hand if i[1] == suits[1])
        if ace_in_suit_1:
            # return second suit if first has ace... Followed the cw tactics
            return suits[1]
        if ace_in_suit_2:
            # return first suit if second suit has ace... Followed the cw tactics
            return suits[0]
```



```
        # This will calculate the total value for the cards in
each suit
        # Add all card values and even the picture cards and ac
e is given a value

        suit1_score = calculate_suit_sum(hand, suits[0])
        suit2_score = calculate_suit_sum(hand, suits[1])
        if suit1_score > suit2_score:
            # If the the first suit has cards with higher value
 then return it
            return suits[0]
        elif suit1_score < suit2_score:
            # If the the second suit has cards with higher valu
e then return it
            return suits[1]
        else:
            # If both suit have cards with equal value then use
            # get the cards for each suit separately
            suit1 = [i for i in hand if i[1] == suits[0]]
            suit2 = [i for i in hand if i[1] == suits[1]]
            # Then I calculated the range of the cards
            # This is an improvement: Reason is
            # eg: It is better to have Q and 7 instead of J and
 8
            # Eben thought they have the same value the first s
et is better
            suit1_range = calculate_suit_range(suit1)
            suit2_range = calculate_suit_range(suit2)

            if suit1_range > suit2_range:
                # So if the values of the cards are far apart f
or suit1 then return the suit as trumps
                return suits[0]
            elif suit1_range < suit2_range:
```
]



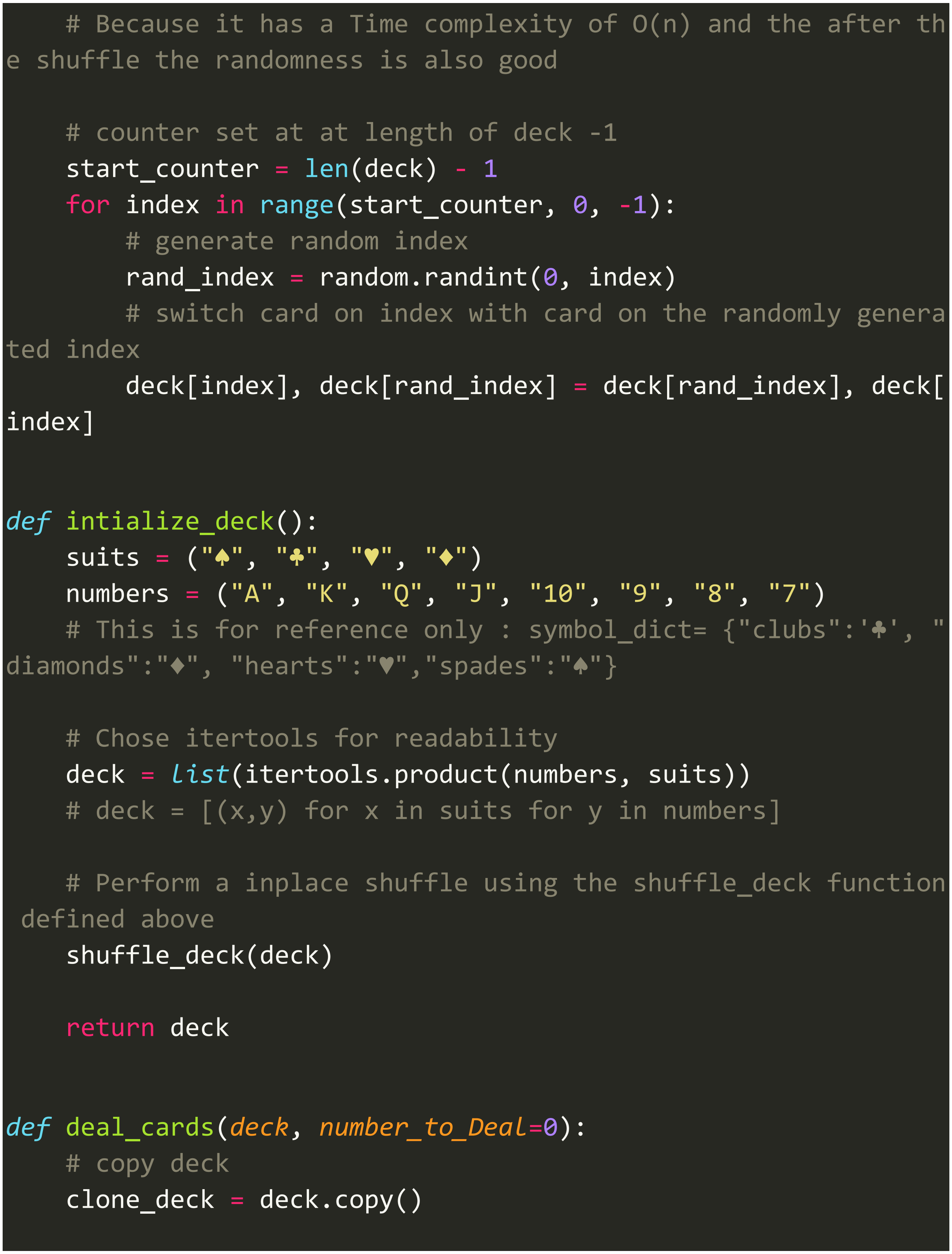
```
                # So if the values of the cards are far apart for suit2 then return the suit as trumps
                return suits[1]

            # Last case the the cards are same for both suits:
            # So used a random choice from random module to select one suit between the two
            return choice(suits)

    elif max_val == 2 and num_max == 1:
        # This when one suit has two cards but the other suits to has one card each
        # Accordind to cw tactics they have told that if the two cards are lower then,
        # The players choose the suit with no cards, hoping to get those cards in the second deal
        # My assumption was if both the two cards were between 7 and 10 incluive then to choose hte other suit

        #  chose the suit with most cards
        max_suit = [i[0] for i in suit_occurence.items() if i[1] == max_val][0]

        #  select all cards which have the max_suit
        max_suit_deck = [i for i in hand if i[1] == max_suit]

        # got the numeric total of the cards, including picture cards and ace
        suit_total = calculate_suit_sum(hand, max_suit)
        # got the averge of the cards
        avg_suit = round(suit_total / 2)
        # got the range of the two cards
        range_suit = calculate_suit_range(max_suit_deck)
        if avg_suit < 10 and range_suit <= 3:
```



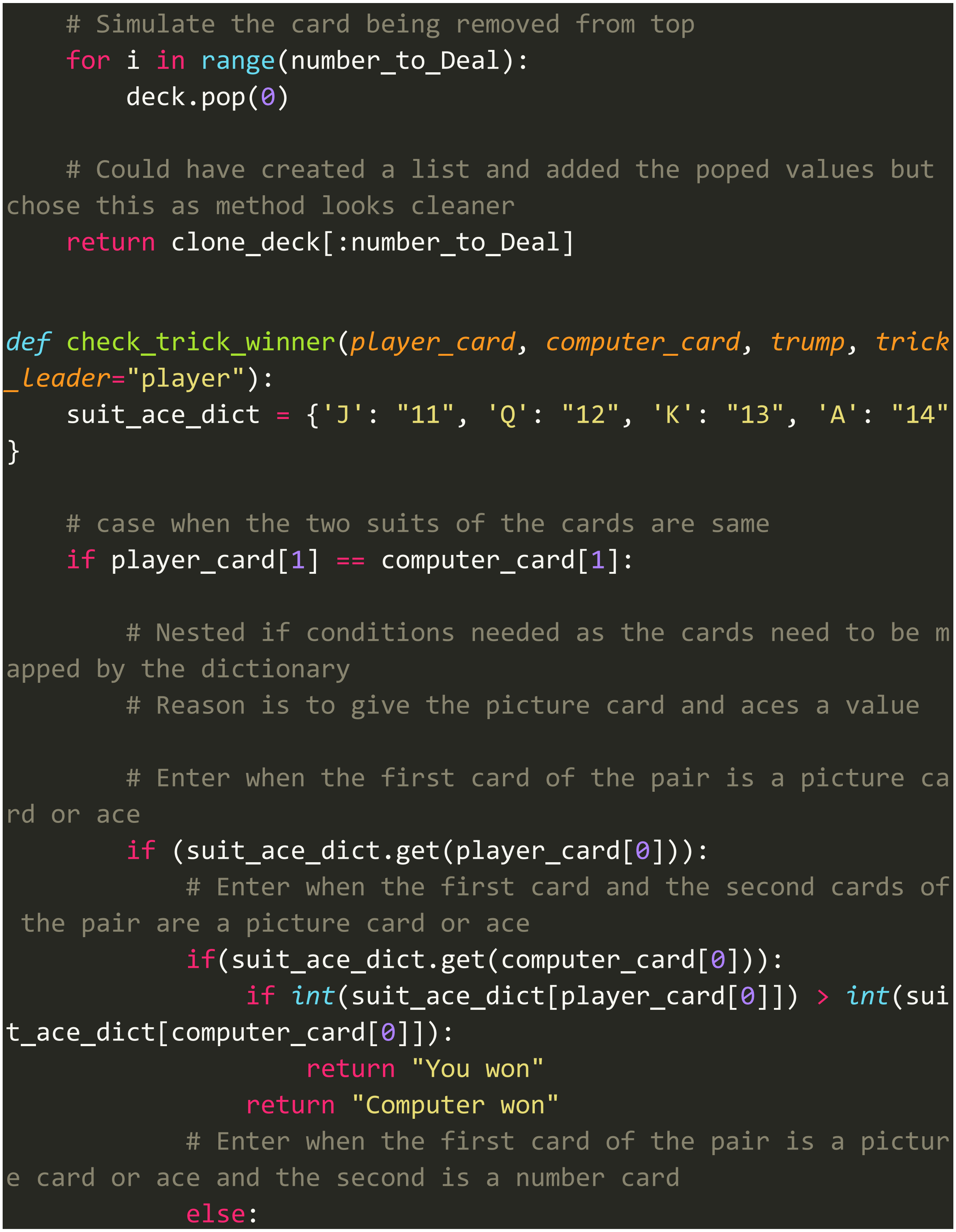
```
            min_suit_deck = [i[0] for i in suit_occurence.items() if i[1] == 0]
            return min_suit_deck[0]
        return max_suit
    else:
        lowest = get_lowest_card(hand)[0]
        lowest_card = [c for c in hand if c[0] == lowest]
        return choice(lowest_card)[1]


def calculate_suit_sum(cards, suit):
    # Helps to implement in finding the sum of all cards
    suit_ace_dict = {'J': "11", 'Q': "12", 'K': "13", 'A': "14"}
    total = 0
    for c in cards:
        if c[1] == suit:
            if (suit_ace_dict.get(c[0])):
                total += int(suit_ace_dict[c[0]])
            else:
                total += int(c[0])
    return total


def calculate_suit_range(deck):
    # calculate the range of given cards include the picture cards and the Aces
    # needed when choosing trumps

    # nested if conditions needed to map a value to picture cards and aces
    suit_ace_dict = {'J': "11", 'Q': "12", 'K': "13", 'A': "14"}
    card1 = deck[0]
    card2 = deck[1]
    if suit_ace_dict.get(card1[0]):
        if suit_ace_dict.get(deck[1][0]):
            return abs(int(suit_ace_dict[card1[0]]) - int(suit_ace_dict[card2[0]]))
        else:
            return abs(int(suit_ace_dict[card1[0]]) - int(card2[0]))
    else:
        if suit_ace_dict.get(card2[0]):
            return abs(int(card1[0]) - int(suit_ace_dict[card2[0]]))
        else:
            return abs(int(deck[0][0]) - int(deck[1][0]))
```

deck.py – Holds all functions mainly connected to the deck of game and the check winner function.

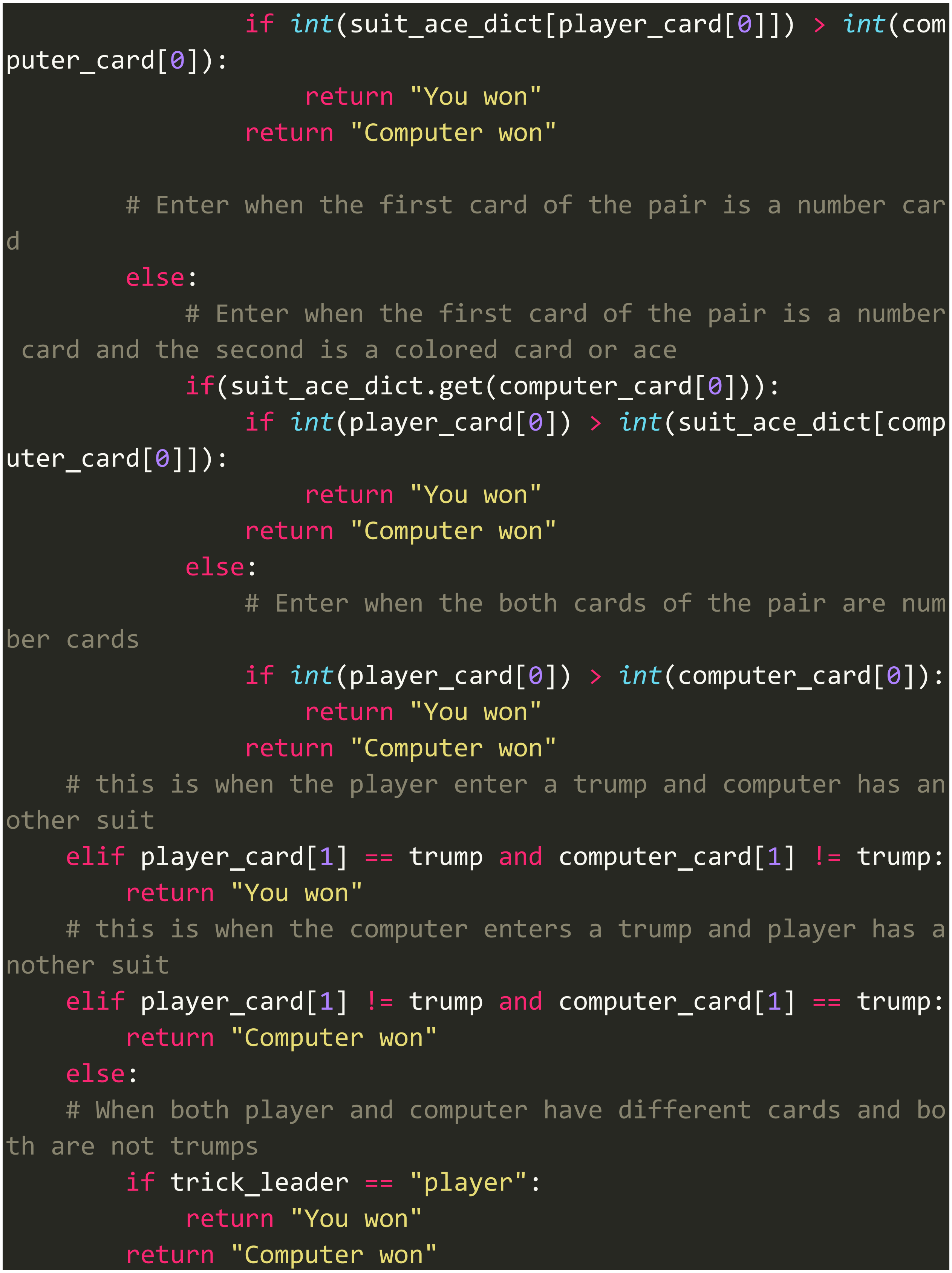
```
"""
This is where the common function involving the deck are stored.

"""

import itertools
import random


def shuffle_deck(deck):
    # Implemented Fisher-Yates algorithm to shuffle the cards
```



```
    # Because it has a Time complexity of O(n) and the after the shuffle the randomness is also good

    # counter set at at length of deck -1
    start_counter = len(deck) - 1
    for index in range(start_counter, 0, -1):
        # generate random index
        rand_index = random.randint(0, index)
        # switch card on index with card on the randomly generated index
        deck[index], deck[rand_index] = deck[rand_index], deck[index]


def intialize_deck():
    suits = ("♠", "♣", "♥", "♦")
    numbers = ("A", "K", "Q", "J", "10", "9", "8", "7")
    # This is for reference only : symbol_dict= {"clubs":'♣', "diamonds":"♦", "hearts":"♥","spades":"♠"}

    # Chose itertools for readability
    deck = list(itertools.product(numbers, suits))
    # deck = [(x,y) for x in suits for y in numbers]

    # Perform a inplace shuffle using the shuffle_deck function defined above
    shuffle_deck(deck)

    return deck


def deal_cards(deck, number_to_Deal=0):
    # copy deck
    clone_deck = deck.copy()
```



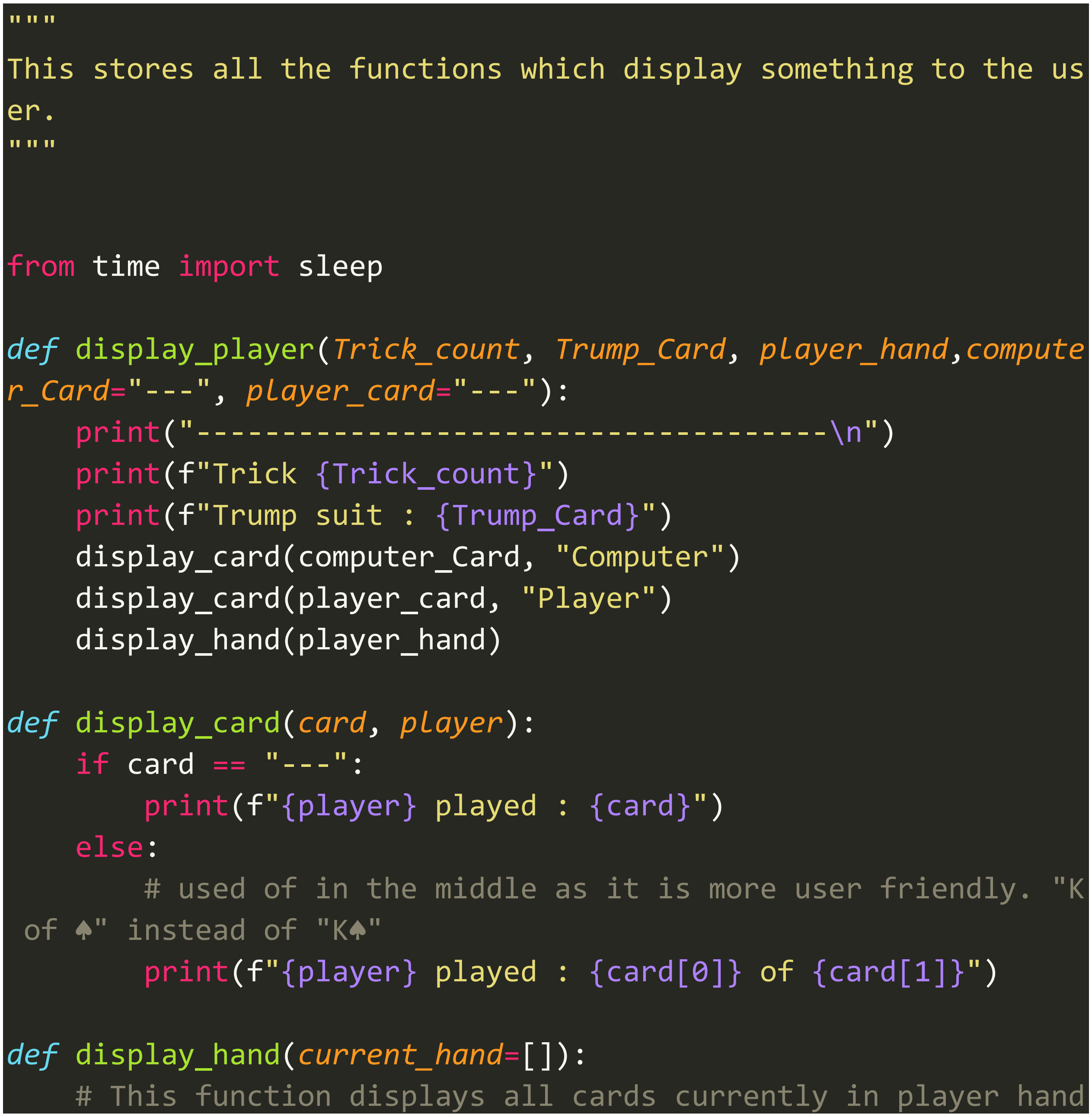
```
    # Simulate the card being removed from top
    for i in range(number_to_Deal):
        deck.pop(0)

    # Could have created a list and added the poped values but
chose this as method looks cleaner
    return clone_deck[:number_to_Deal]


def check_trick_winner(player_card, computer_card, trump, trick
_leader="player"):
    suit_ace_dict = {'J': "11", 'Q': "12", 'K': "13", 'A': "14"
}

    # case when the two suits of the cards are same
    if player_card[1] == computer_card[1]:

        # Nested if conditions needed as the cards need to be m
apped by the dictionary
        # Reason is to give the picture card and aces a value

        # Enter when the first card of the pair is a picture ca
rd or ace
        if (suit_ace_dict.get(player_card[0])):
            # Enter when the first card and the second cards of
 the pair are a picture card or ace
            if(suit_ace_dict.get(computer_card[0])):
                if int(suit_ace_dict[player_card[0]]) > int(sui
t_ace_dict[computer_card[0]]):
                    return "You won"
                return "Computer won"
            # Enter when the first card of the pair is a pictur
e card or ace and the second is a number card
            else:
```



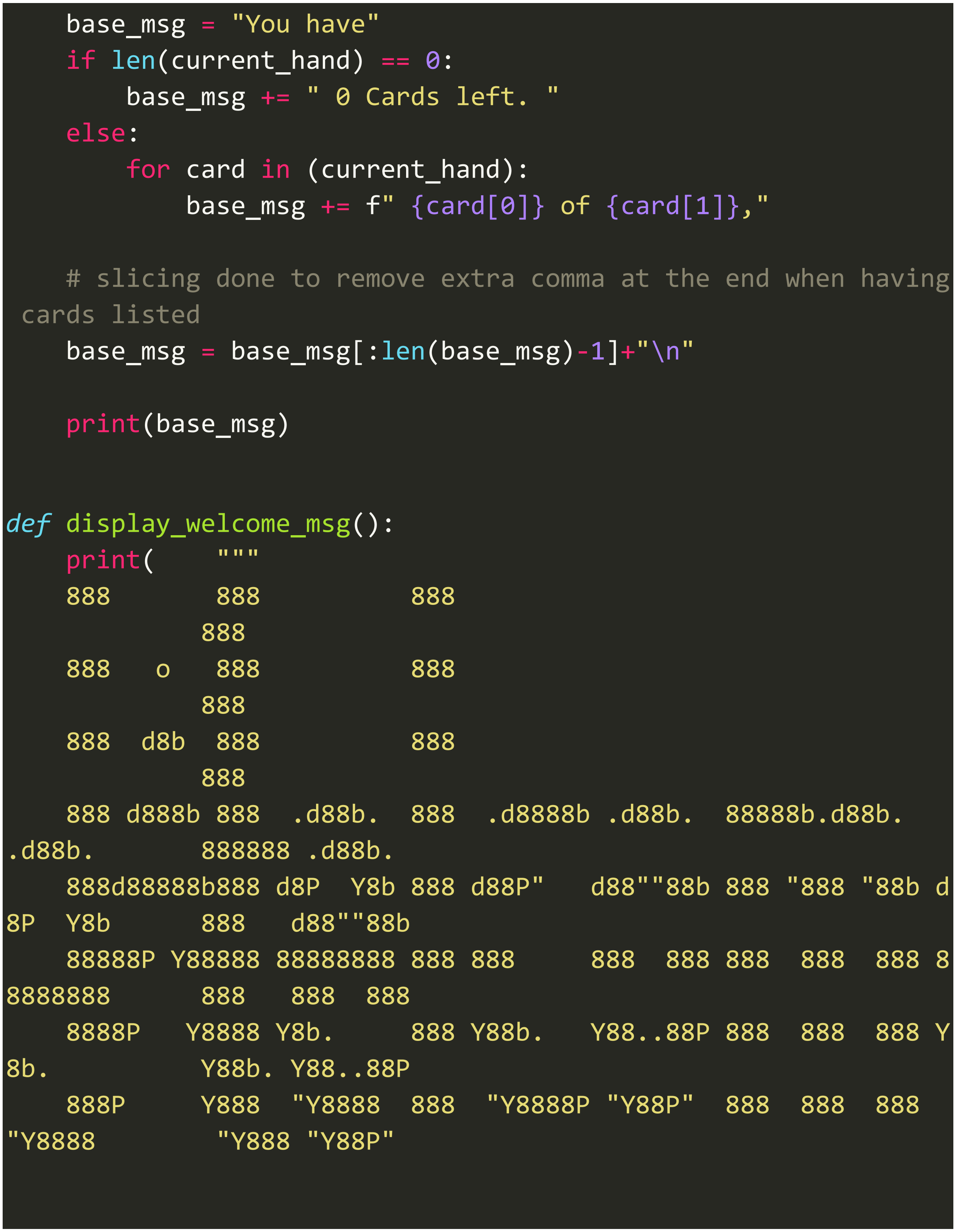
```
                if int(suit_ace_dict[player_card[0]]) > int(computer_card[0]):
                    return "You won"
                return "Computer won"

        # Enter when the first card of the pair is a number card
        else:
            # Enter when the first card of the pair is a number card and the second is a colored card or ace
            if(suit_ace_dict.get(computer_card[0])):
                if int(player_card[0]) > int(suit_ace_dict[computer_card[0]]):
                    return "You won"
                return "Computer won"
            else:
                # Enter when the both cards of the pair are number cards
                if int(player_card[0]) > int(computer_card[0]):
                    return "You won"
                return "Computer won"
    # this is when the player enter a trump and computer has another suit
    elif player_card[1] == trump and computer_card[1] != trump:
        return "You won"
    # this is when the computer enters a trump and player has another suit
    elif player_card[1] != trump and computer_card[1] == trump:
        return "Computer won"
    else:
    # When both player and computer have different cards and both are not trumps
        if trick_leader == "player":
            return "You won"
        return "Computer won"
```

```
"""
try:
    case1 = check_trick_winner(("K", "♥"), ("9", "♥"), "♠", "co
mputer")
    assert case1 == "You won", "Case 1 failed"

    case2 = check_trick_winner(("8", "♦"), ("9", "♦"), "♦", "pl
ayer")
    assert case2 == "Computer won", "Case 2 failed"

    case3 = check_trick_winner(("10", "♣"), ("J", "♥"), "♣", "c
omputer")
    assert case3 == "You won", "Case 3 failed"

    case4 = check_trick_winner(("Q", "♦"), ("9", "♥"), "♥", "co
mputer")
    assert case4 == "Computer won", "Case 4 failed"

    case5 = check_trick_winner(("Q", "♠"), ("A", "♦"), "♥", "pl
ayer")
    assert case5 == "You won", "Case 5 failed"

    case6 = check_trick_winner(("10", "♦"), ("10", "♠"), "♣", "
computer")
    assert case6 == "Computer won", "Case 6 failed"

except AssertionError as e:
    print(e)

else:
    print("All test cases pass")
    # All test cases pass
"""
```

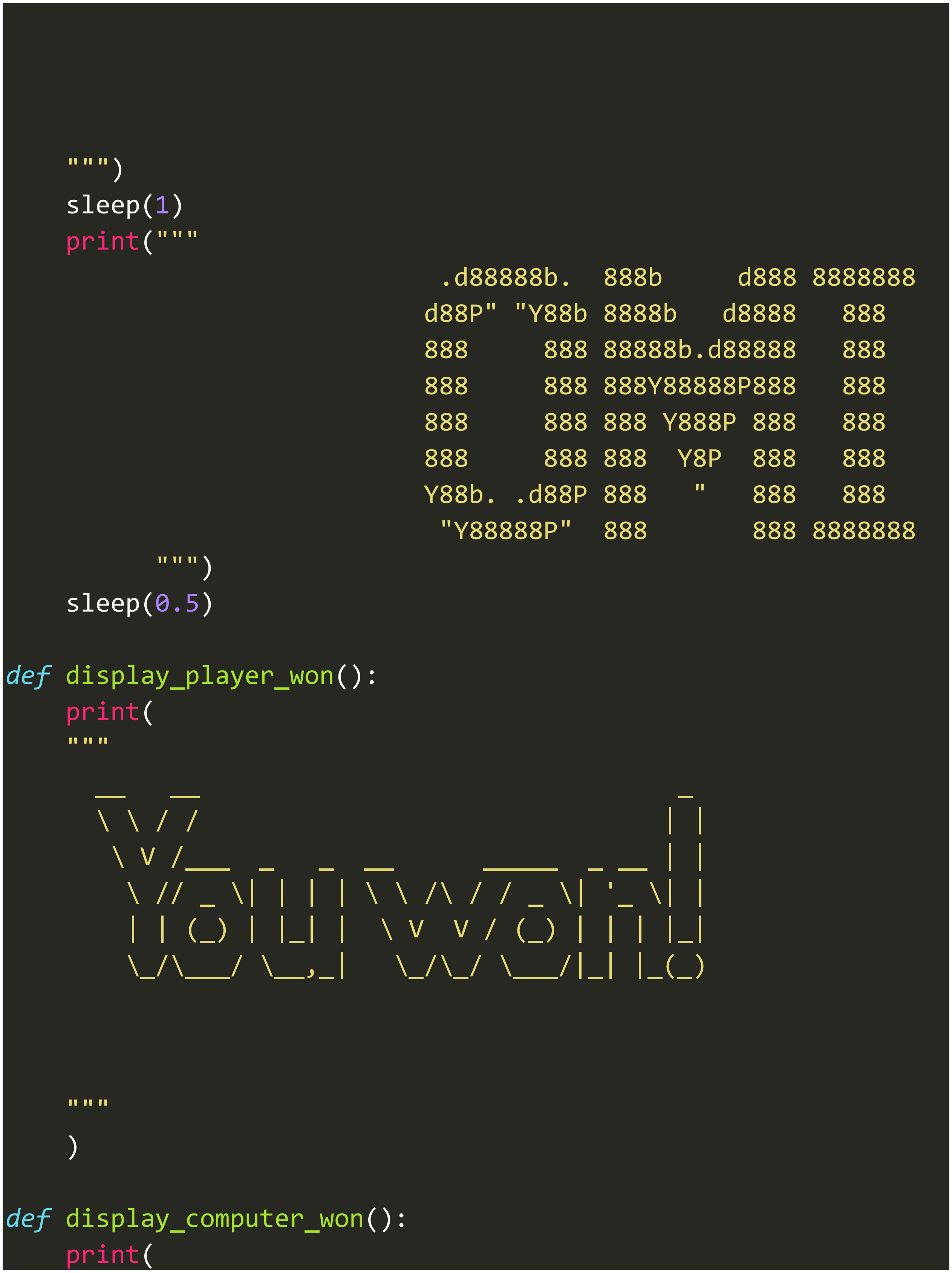
display_func.py – Stores all function, that display content to user.



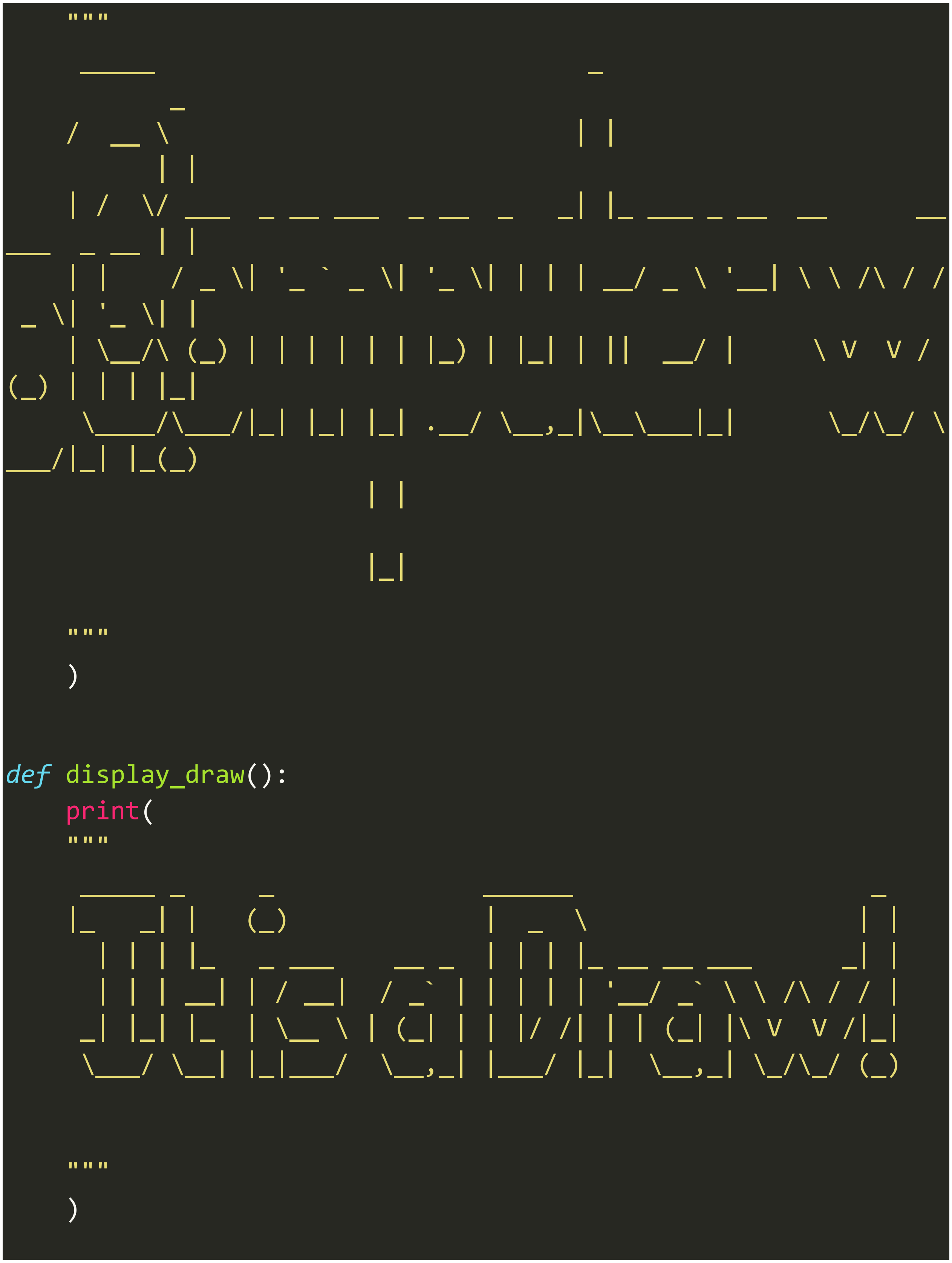
```
"""
This stores all the functions which display something to the user.
"""


from time import sleep

def display_player(Trick_count, Trump_Card, player_hand,computer_Card="---", player_card="---"):
    print("-------------------------------------\n")
    print(f"Trick {Trick_count}")
    print(f"Trump suit : {Trump_Card}")
    display_card(computer_Card, "Computer")
    display_card(player_card, "Player")
    display_hand(player_hand)

def display_card(card, player):
    if card == "---":
        print(f"{player} played : {card}")
    else:
        # used of in the middle as it is more user friendly. "K of ♠" instead of "K♠"
        print(f"{player} played : {card[0]} of {card[1]}")

def display_hand(current_hand=[]):
    # This function displays all cards currently in player hand
```



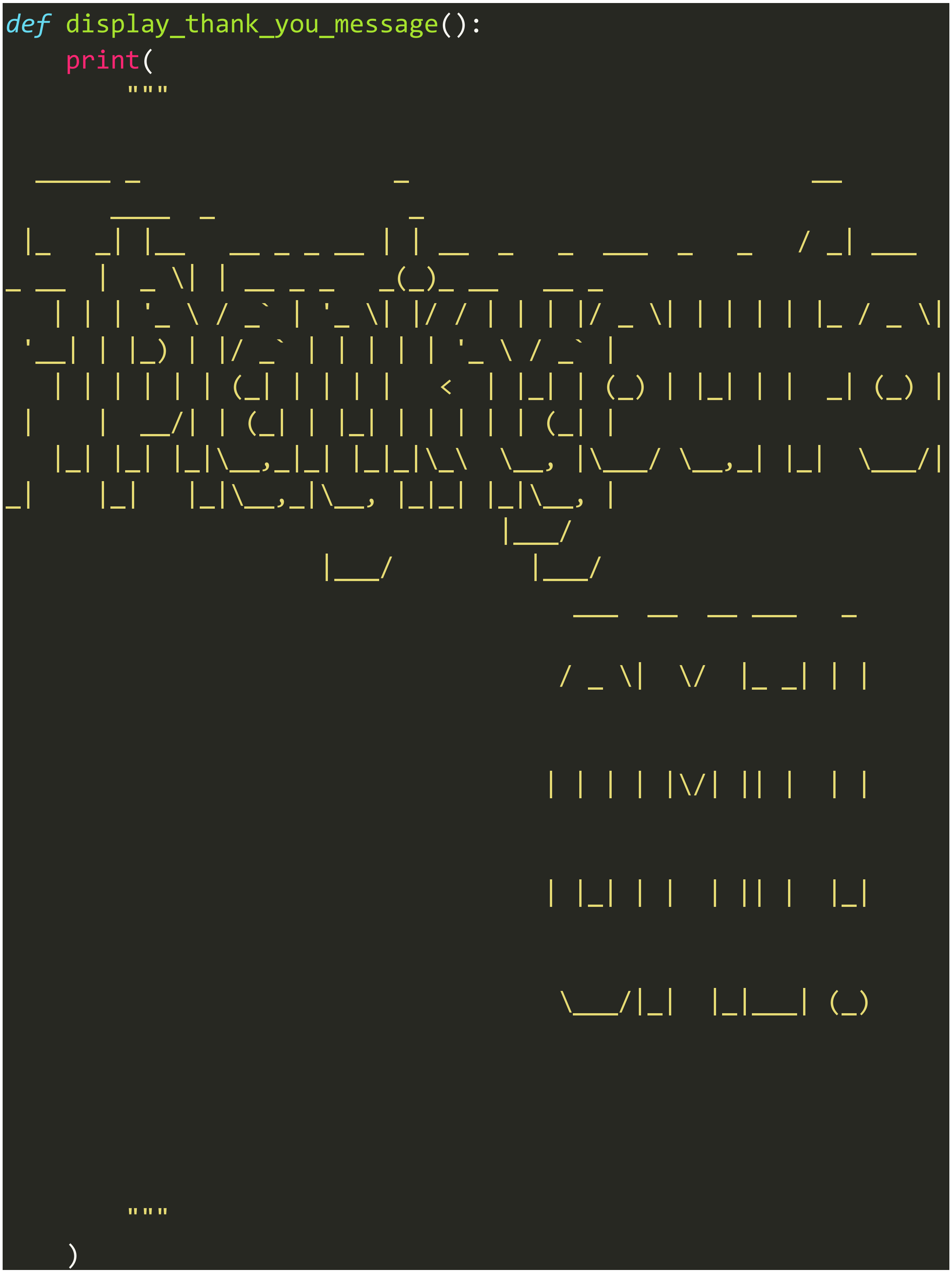
```
    base_msg = "You have"
    if len(current_hand) == 0:
        base_msg += " 0 Cards left. "
    else:
        for card in (current_hand):
            base_msg += f" {card[0]} of {card[1]},"

    # slicing done to remove extra comma at the end when having
 cards listed
    base_msg = base_msg[:len(base_msg)-1]+"\n"

    print(base_msg)


def display_welcome_msg():
    print(    """
    888       888          888
             888
    888   o   888          888
             888
    888  d8b  888          888
             888
    888 d8888b 888  .d88b.  888  .d8888b .d88b.  88888b.d88b.
.d88b.       888888 .d88b.
    888d88888b888 d8P  Y8b 888 d8P"   d88""88b 888 "888 "88b d
8P  Y8b      888   d88""88b
    88888P Y88888 88888888 888 888     888  888 888  888  888 8
8888888      888   888  888
    8888P   Y8888 Y8b.     888 Y88b.   Y88..88P 888  888  888 Y
8b.          Y88b. Y88..88P
    888P     Y888  "Y8888  888  "Y8888P "Y88P"  888  888  888
"Y8888        "Y888 "Y88P"
```



```
    """)
    sleep(1)
    print("""
                             .d88888b.  888b     d888 8888888
                            d88P" "Y88b 8888b   d8888   888
                            888     888 88888b.d88888   888
                            888     888 888Y88888P888   888
                            888     888 888 Y888P 888   888
                            888     888 888  Y8P  888   888
                            Y88b. .d88P 888   "   888   888
                             "Y88888P"  888       888 8888888
          """)
    sleep(0.5)

def display_player_won():
    print(
    """
      __   __                               _
      \ \ / /                              | |
       \ V /___  _   _  __      _____  _ __ | |
        \ // _ \| | | | \ \ /\ / / _ \| '_ \| |
        | | (_) | |_| |  \ V  V / (_) | | | |_|
        \_/\___/ \__,_|   \_/\_/ \___/|_| |_(_)



    """
    )

def display_computer_won():
    print(
```



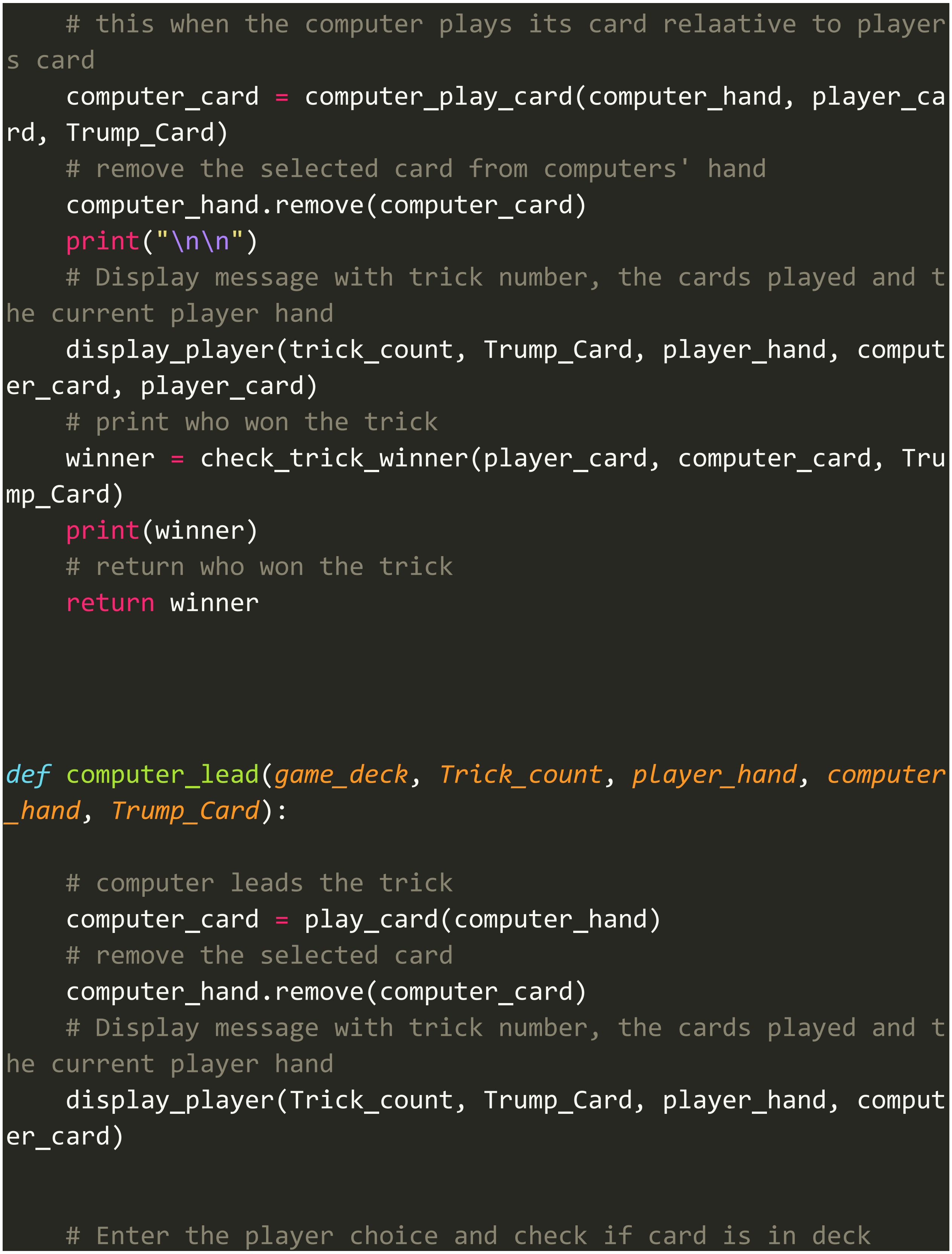
```
    """
     _____                                 _
           _
    /  __ \                           | |
          | |
    | /  \/ ___  _ __ ___  _ __  _   _| |_ ___ _ __  __      __
___  _ __ | |
    | |    / _ \| '_ ` _ \| '_ \| | | | __/ _ \ '__| \ \ /\ / /
 _ \| '_ \| |
    | \__/\ (_) | | | | | | |_) | |_| | ||  __/ |     \ V  V /
(_) | | | |_|
     \____/\___/|_| |_| |_| .__/ \__,_|\__\___|_|      \_/\_/ \
___/|_| |_(_)
                        | |

                        |_|

    """
    )


def display_draw():
    print(
    """
     _____ _     _              ______                    _
    |_   _| |   (_)             |  _  \                  | |
      | | | |_   _ ___    __ _  | | | |_ __ __ ___      _| |
      | | | __| | / __|  / _` | | | | | '__/ _` \ \ /\ / / |
     _| |_| |_  | \__ \ | (_| | | |/ /| | | (_| |\ V  V /|_|
     \___/ \__| |_|___/  \__,_| |___/ |_|  \__,_| \_/\_/ (_)


    """
    )
```



```
def display_thank_you_message():
    print(
        """


  _____ _                 _                         __
     ____  _            _
 |_   _| |__   __ _ _ __ | | __  _   _  ___  _   _   / _| ___
_ __  |  _ \| | __ _ _   _(_)_ __   __ _
   | | | '_ \ / _` | '_ \| |/ / | | | |/ _ \| | | | | |_ / _ \|
 '__| | |_) | |/ _` | | | | | '_ \ / _` |
   | | | | | | (_| | | | |   <  | |_| | (_) | |_| | |  _| (_) |
 |    |  __/| | (_| | |_| | | | | | (_| |
   |_| |_| |_|\__,_|_| |_|_|\_\  \__, |\___/ \__,_| |_|  \___/|
_|    |_|   |_|\__,_|\__, |_|_| |_|\__, |
                                  |___/
                     |___/        |___/
                                   ___  __  __ ___   _

                                  / _ \|  \/  |_ _| | |

                                 | | | | |\/| || |  | |

                                 | |_| | |  | || |  |_|

                                  \___/|_|  |_|___| (_)




        """
    )
```

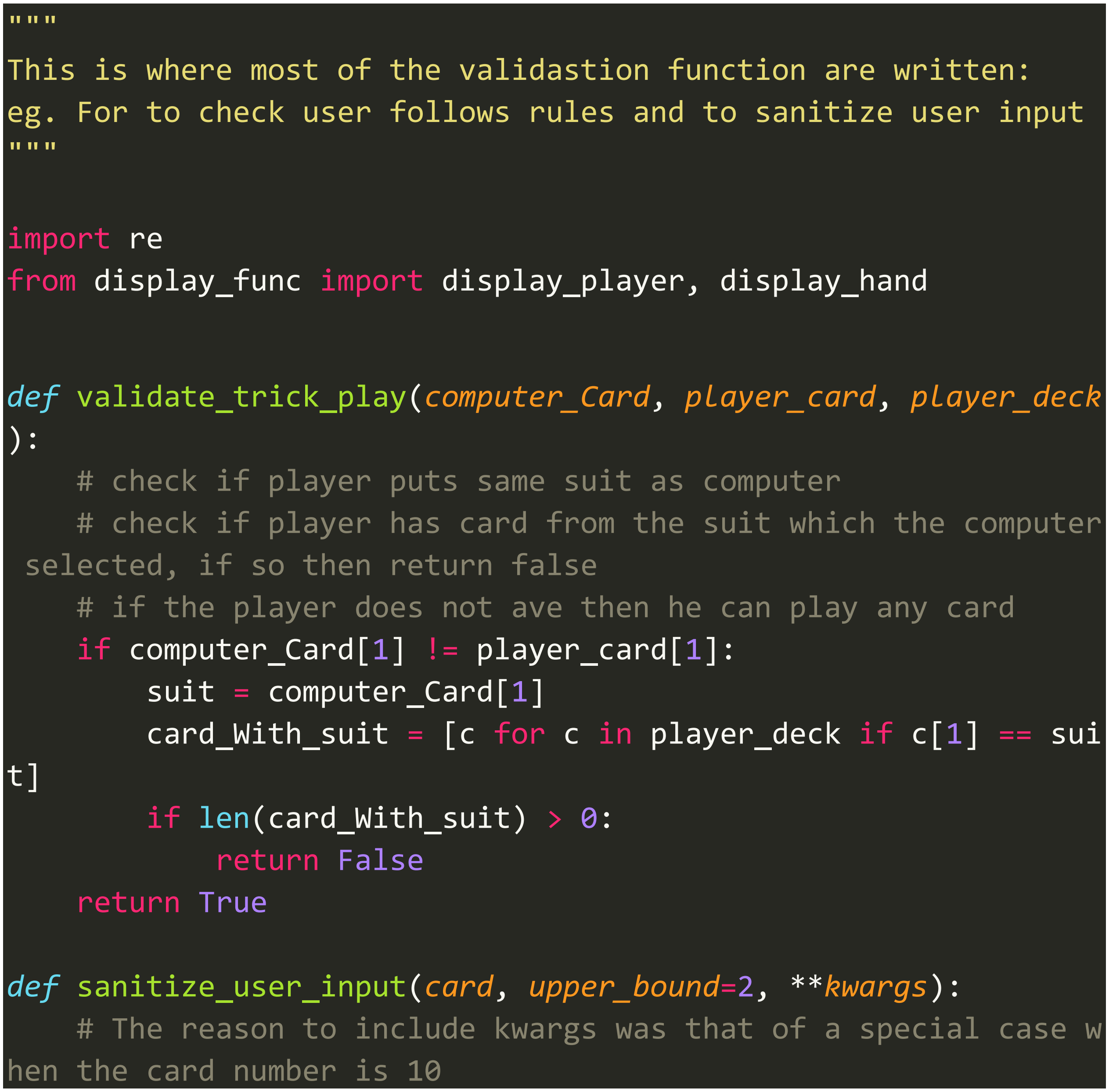
Game_logic.py – It is where the majority of the game phase is implemented.

```
"""
This is where the game logic is avilable dor each scenario:
1) When the computer leads
2) When the player leads

"""

from deck import check_trick_winner
from validate_func import validate_for_Card_in_Deck, validate_trump_suit, validate_trick_play
from display_func import display_hand, display_welcome_msg, display_player
from computer import computer_play_card, get_lowest_card, choose_trump, play_card


# This is when player leads a trick
def player_lead(game_deck, trick_count, player_hand, computer_hand, Trump_Card):

    # This is when the player enters the card and it is checked whether card is in player's hand
    player_card = validate_for_Card_in_Deck(trick_count, Trump_Card, player_hand)
    # remove the selected card from players hand
    player_hand.remove(player_card)
```

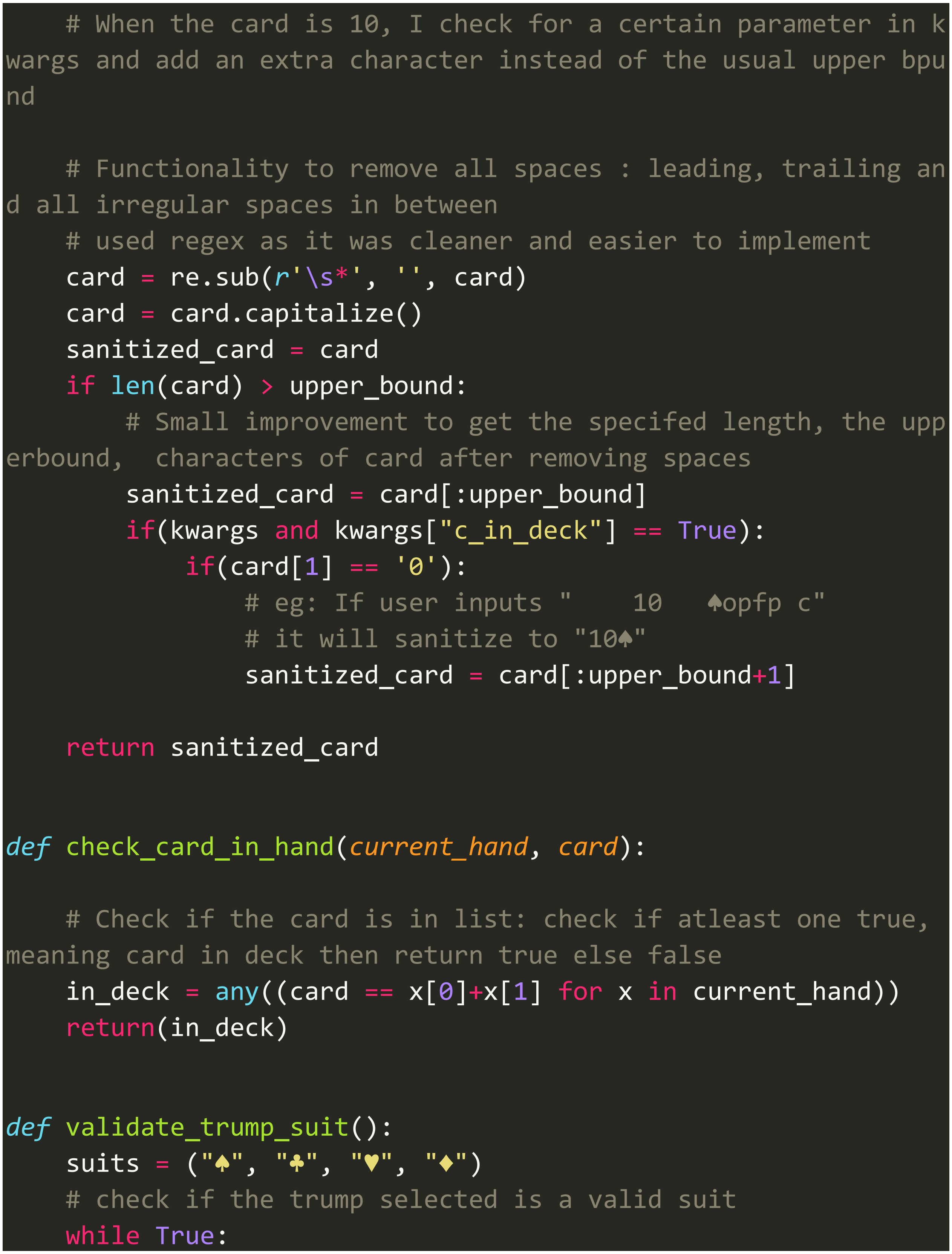


```
    # this when the computer plays its card relaative to players card
    computer_card = computer_play_card(computer_hand, player_card, Trump_Card)
    # remove the selected card from computers' hand
    computer_hand.remove(computer_card)
    print("\n\n")
    # Display message with trick number, the cards played and the current player hand
    display_player(trick_count, Trump_Card, player_hand, computer_card, player_card)
    # print who won the trick
    winner = check_trick_winner(player_card, computer_card, Trump_Card)
    print(winner)
    # return who won the trick
    return winner




def computer_lead(game_deck, Trick_count, player_hand, computer_hand, Trump_Card):

    # computer leads the trick
    computer_card = play_card(computer_hand)
    # remove the selected card
    computer_hand.remove(computer_card)
    # Display message with trick number, the cards played and the current player hand
    display_player(Trick_count, Trump_Card, player_hand, computer_card)


    # Enter the player choice and check if card is in deck
    player_card = validate_for_Card_in_Deck(Trick_count, Trump_Card, player_hand, computer_card)
    # check whether the player followed the OMI rules
    valid = validate_trick_play(computer_card, player_card, player_hand)
    # below repeats above process until all conditions are met
    while not valid:
        display_player(Trick_count, Trump_Card, player_hand, computer_card)
        print(f"Follow the rules, cannot put a card of different suit if you have same suit.\nPlay card with suit {computer_card[1]}")
        player_card = validate_for_Card_in_Deck(Trick_count, Trump_Card, player_hand, computer_card)
        valid = validate_trick_play(computer_card, player_card, player_hand)
    # remove the card the player played
    player_hand.remove(player_card)


    # Display message with trick number, the cards played and the current player hand
    display_player(Trick_count, Trump_Card, player_hand, computer_card, player_card)
    # print who won the trick
    winner = check_trick_winner(player_card, computer_card, Trump_Card, "computer")
    print(winner)
    # return who won the trick
    return winner
```

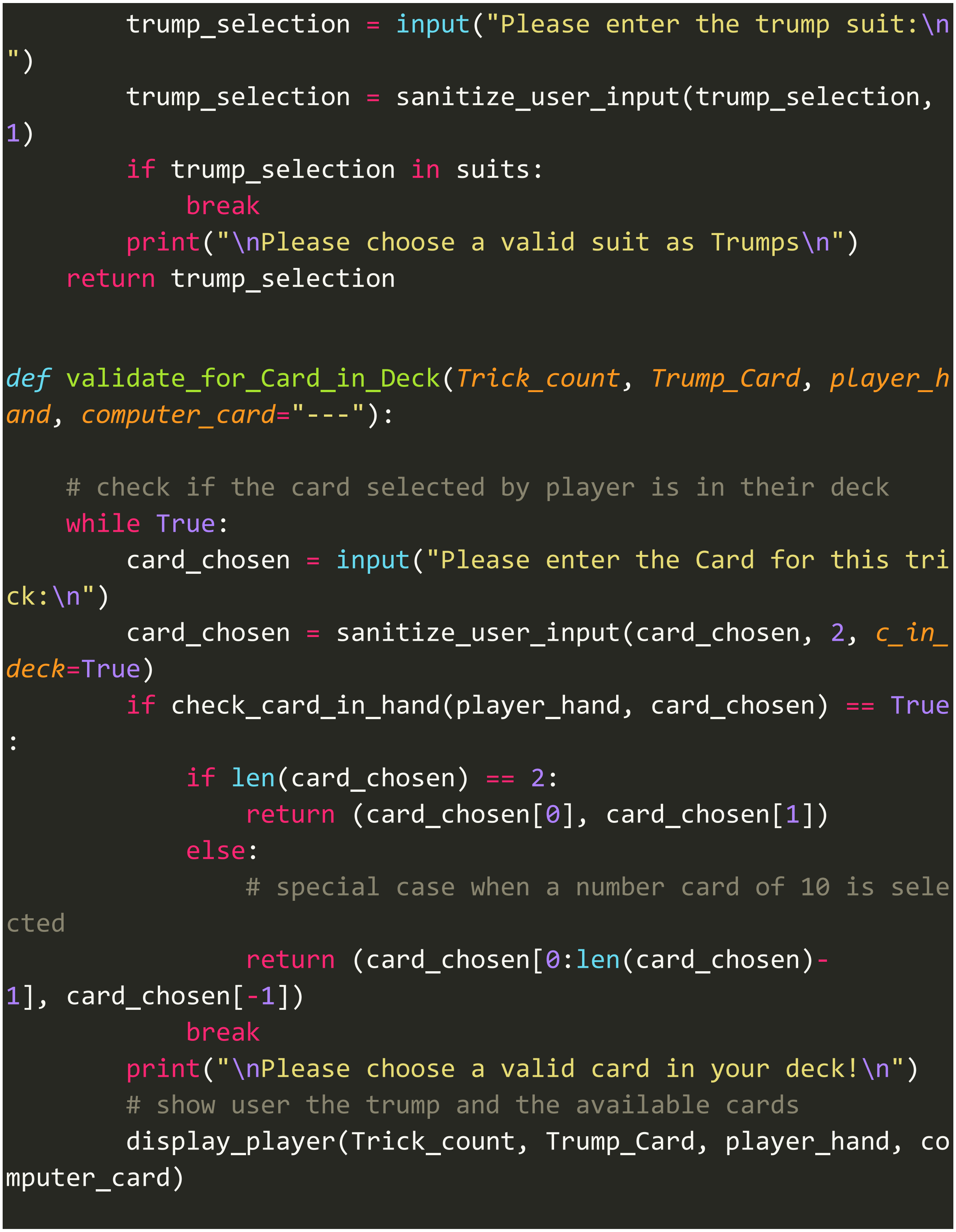
validate_func.py – Stores all function related to validation and sanitization.



```
"""
This is where most of the validastion function are written:
eg. For to check user follows rules and to sanitize user input
"""

import re
from display_func import display_player, display_hand


def validate_trick_play(computer_Card, player_card, player_deck
):
    # check if player puts same suit as computer
    # check if player has card from the suit which the computer
 selected, if so then return false
    # if the player does not ave then he can play any card
    if computer_Card[1] != player_card[1]:
        suit = computer_Card[1]
        card_With_suit = [c for c in player_deck if c[1] == sui
t]
        if len(card_With_suit) > 0:
            return False
    return True

def sanitize_user_input(card, upper_bound=2, **kwargs):
    # The reason to include kwargs was that of a special case w
hen the card number is 10
```



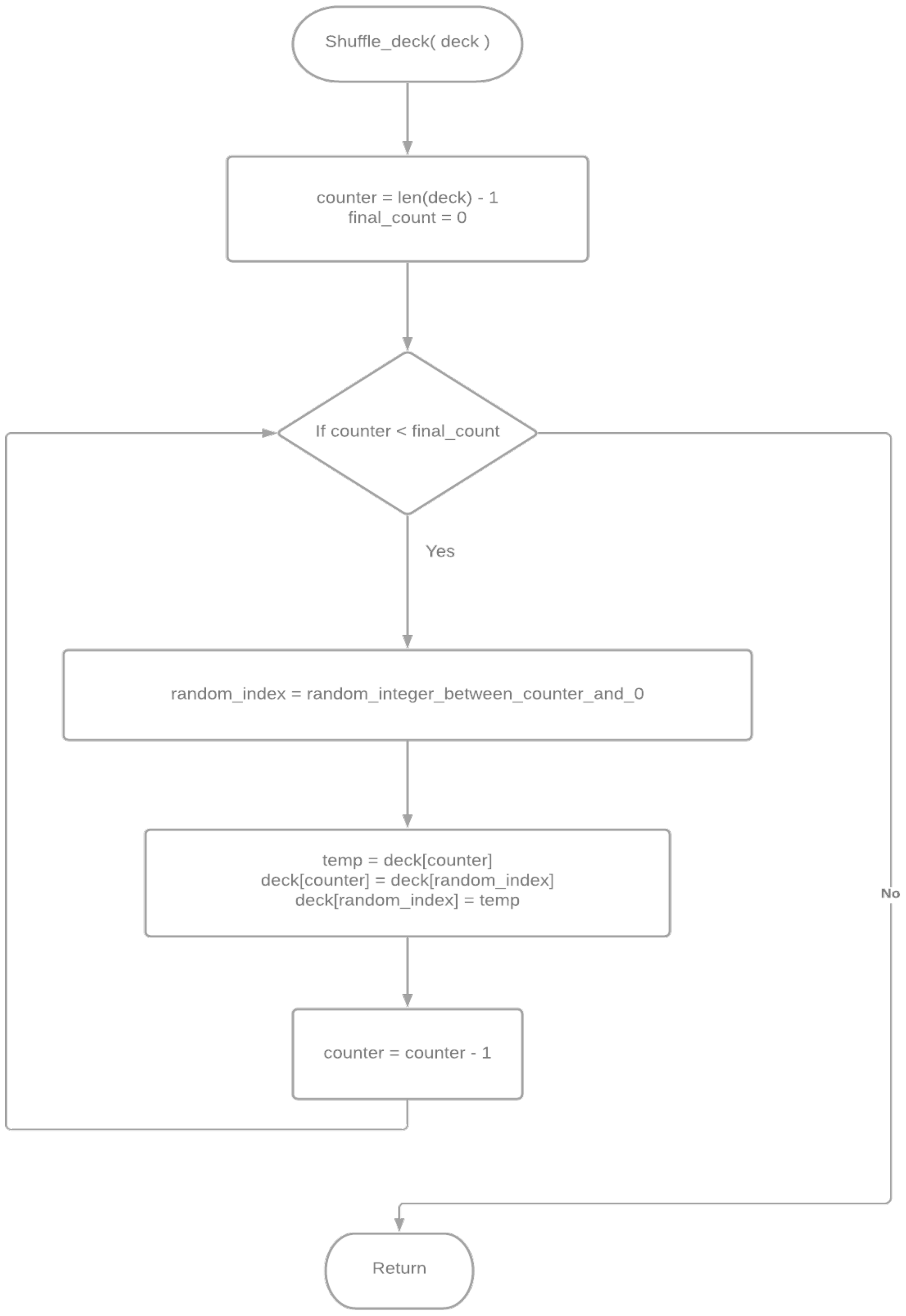
```
    # When the card is 10, I check for a certain parameter in kwargs and add an extra character instead of the usual upper bpund

    # Functionality to remove all spaces : leading, trailing and all irregular spaces in between
    # used regex as it was cleaner and easier to implement
    card = re.sub(r'\s*', '', card)
    card = card.capitalize()
    sanitized_card = card
    if len(card) > upper_bound:
        # Small improvement to get the specifed length, the upperbound,  characters of card after removing spaces
        sanitized_card = card[:upper_bound]
        if(kwargs and kwargs["c_in_deck"] == True):
            if(card[1] == '0'):
                # eg: If user inputs "    10   ♠opfp c"
                # it will sanitize to "10♠"
                sanitized_card = card[:upper_bound+1]

    return sanitized_card


def check_card_in_hand(current_hand, card):

    # Check if the card is in list: check if atleast one true, meaning card in deck then return true else false
    in_deck = any((card == x[0]+x[1] for x in current_hand))
    return(in_deck)


def validate_trump_suit():
    suits = ("♠", "♣", "♥", "♦")
    # check if the trump selected is a valid suit
    while True:
```



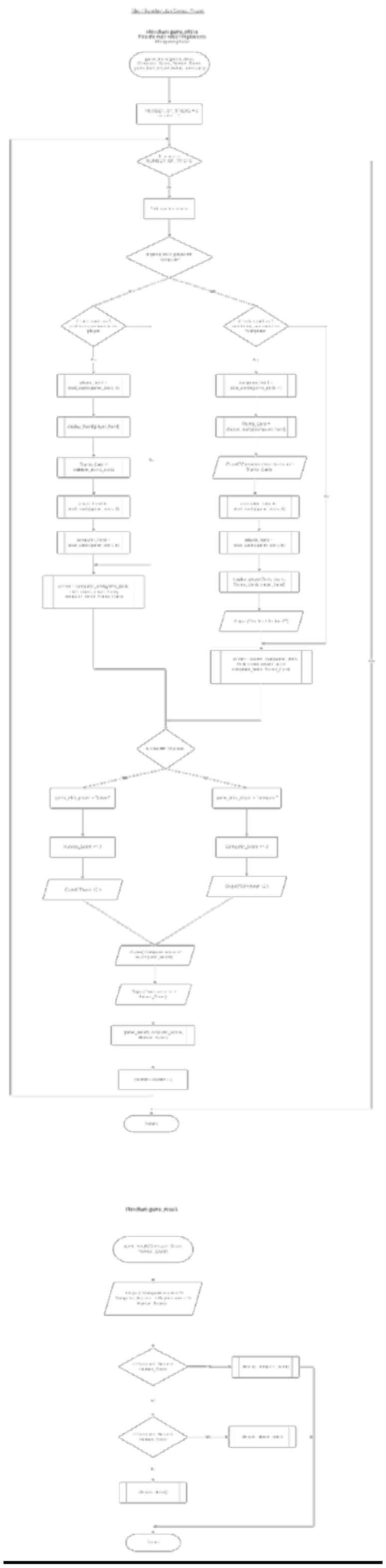
```python
        trump_selection = input("Please enter the trump suit:\n")
        trump_selection = sanitize_user_input(trump_selection, 1)
        if trump_selection in suits:
            break
        print("\nPlease choose a valid suit as Trumps\n")
    return trump_selection


def validate_for_Card_in_Deck(Trick_count, Trump_Card, player_hand, computer_card="---"):

    # check if the card selected by player is in their deck
    while True:
        card_chosen = input("Please enter the Card for this trick:\n")
        card_chosen = sanitize_user_input(card_chosen, 2, c_in_deck=True)
        if check_card_in_hand(player_hand, card_chosen) == True:
            if len(card_chosen) == 2:
                return (card_chosen[0], card_chosen[1])
            else:
                # special case when a number card of 10 is selected
                return (card_chosen[0:len(card_chosen)-1], card_chosen[-1])
            break
        print("\nPlease choose a valid card in your deck!\n")
        # show user the trump and the available cards
        display_player(Trick_count, Trump_Card, player_hand, computer_card)
```

[redacted]

The shuffling algorithm Flowchart

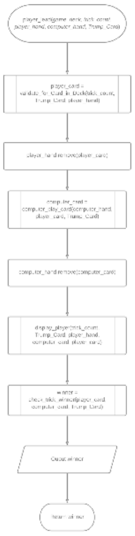


Game Phase Flowcharts

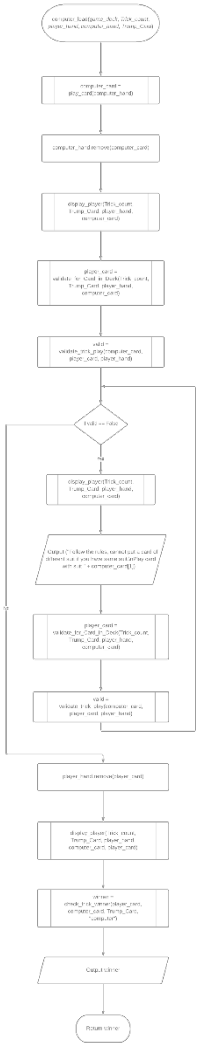


Below are the main two game logic functions

Flowchart: player_lead

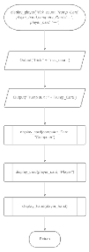

player_lead(game_deck, trick_count, player_hand, computer_hand, Trump_Card)
player_card = validate_for_Card_in_Deck(trick_count, Trump_Card, player_hand)
player_hand.remove(player_card)
computer_card = computer_play_card(computer_hand, player_card, Trump_Card)
computer_hand.remove(computer_card)
display_player(trick_count, Trump_Card, player_hand, computer_card, player_card)
winner = check_trick_winner(player_card, computer_card, Trump_Card)
Output winner
Return winner

Flowchart: computer_lead

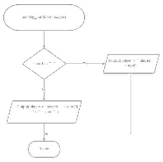

computer_lead(game_deck, Trick_count, player_hand, computer_hand, Trump_Card)
computer_card = play_card(computer_hand)
computer_hand.remove(computer_card)
display_player(Trick_count, Trump_Card, player_hand, computer_card)
player_card = validate_for_Card_in_Deck(Trick_count, Trump_Card, player_hand, computer_card)
valid = validate_trick_play(computer_card, player_card, player_hand)
If valid == False
Yes
No
display_player(Trick_count, Trump_Card, player_hand, computer_card)
Output ("Follow the rules, cannot put a card of different suit if you have same suit.\nPlay card with suit " + computer_card[1])
player_card = validate_for_Card_In_Deck(Trick_count, Trump_Card, player_hand, computer_card)
valid = validate_trick_play(computer_card, player_card, player_hand)
player_hand.remove(player_card)
display_player(Trick_count, Trump_Card, player_hand, computer_card, player_card)
winner = check_trick_winner(player_card, computer_card, Trump_Card, "computer")
Output winner
Return winner

Flowchart for Display functions
[illegible]

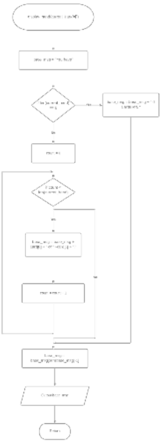
Flowchart display_player



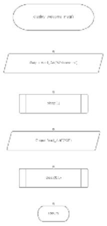
Flowchart display_card



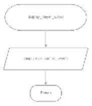
Flowchart display_hand



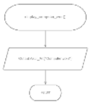
Flowchart display_welcome_msg



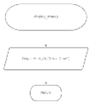
Flowchart display_player_won

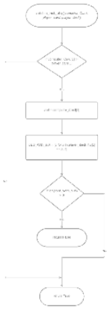


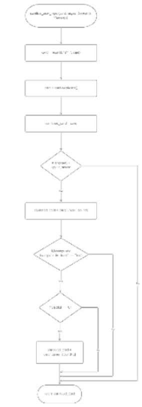
Flowchart display_computer_won

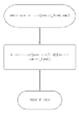


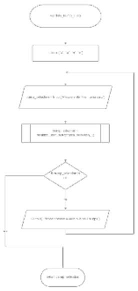
Flowchart: display_draw

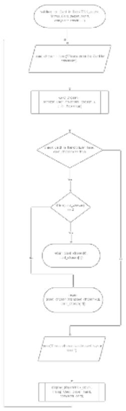






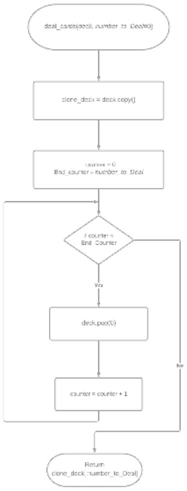




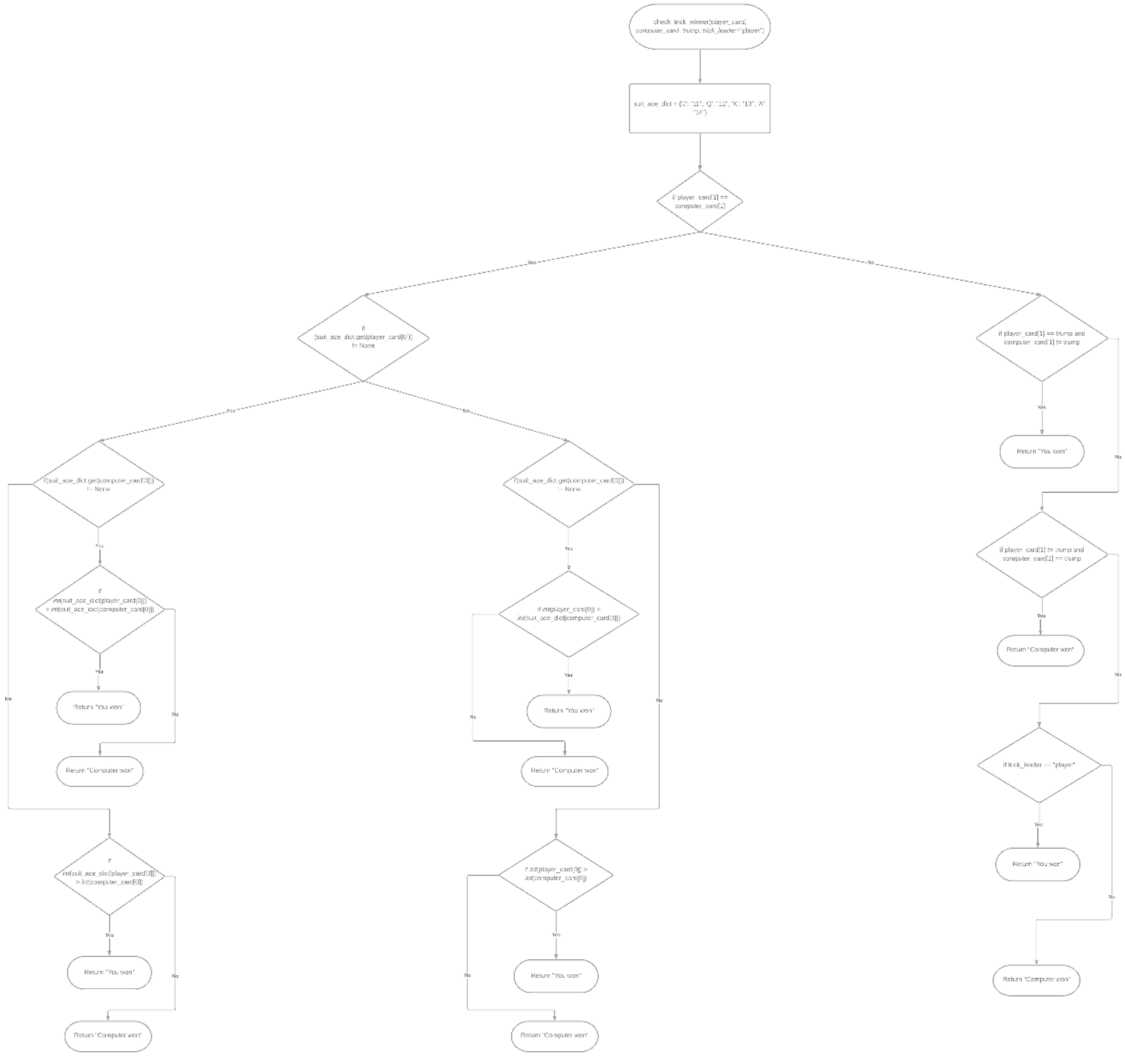


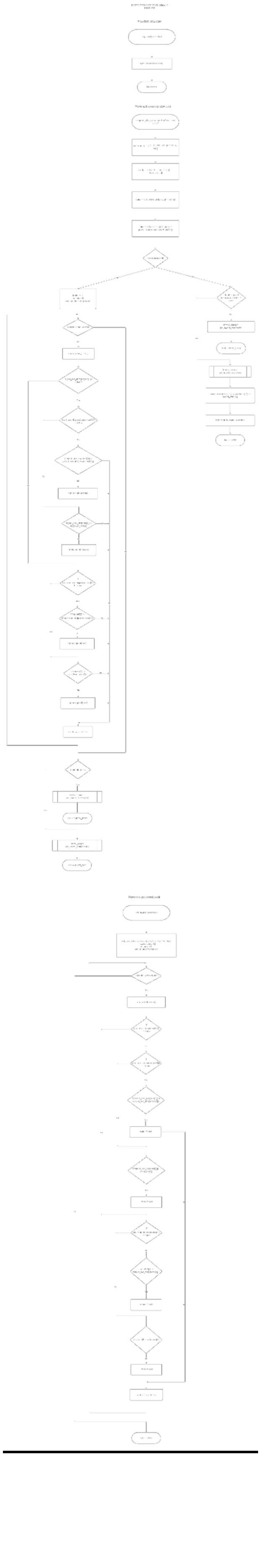
Below are the Deck functions.

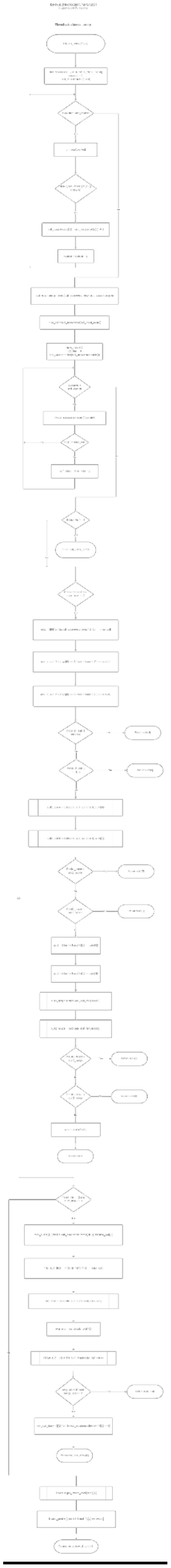
Flowchart: deal_cards



Flowchart: check_trick_winner

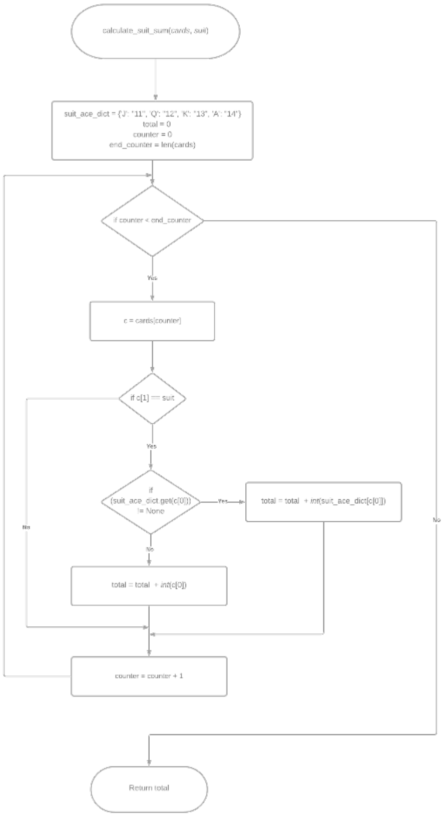

check_trick_winner(player_card,
computer_card, trump, trick_leader="player")
suit_ace_dict = {'J': "11", 'Q': "12", 'K': "13", 'A': "14"}
if player_card[1] ==
computer_card[1]
Yes
No
if
(suit_ace_dict.get(player_card[0]))
!= None
Yes
No
if(suit_ace_dict.get(computer_card[0]))
!= None
Yes
if
int(suit_ace_dict[player_card[0]])
> int(suit_ace_dict[computer_card[0]])
Yes
Return "You won"
No
Return "Computer won"
No
if
int(suit_ace_dict[player_card[0]])
> int(computer_card[0])
Yes
Return "You won"
No
Return "Computer won"
if(suit_ace_dict.get(computer_card[0]))
!= None
Yes
if int(player_card[0]) >
int(suit_ace_dict[computer_card[0]])
Yes
Return "You won"
No
Return "Computer won"
No
if int(player_card[0]) >
int(computer_card[0])
Yes
Return "You won"
No
Return "Computer won"
if player_card[1] == trump and
computer_card[1] != trump
Yes
Return "You won"
No
if player_card[1] != trump and
computer_card[1] == trump
Yes
Return "Computer won"
No
if trick_leader == "player"
Yes
Return "You won"
No
Return "Computer won"



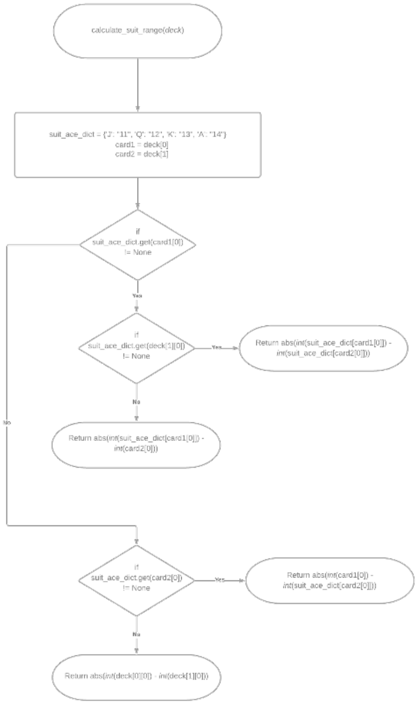


These are the final set of function in Computer Functions.

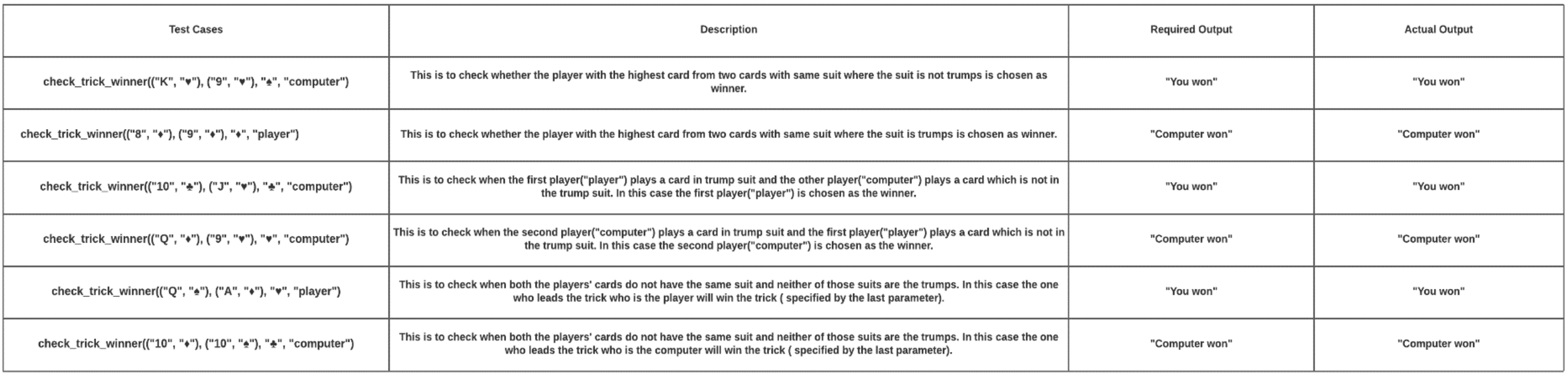
Flowchart: calculate_suit_sum


calculate_suit_sum(*cards*, *suit*)
suit_ace_dict = {'J': "11", 'Q': "12", 'K': "13", 'A': "14"}
total = 0
counter = 0
end_counter = len(cards)
if counter < end_counter
Yes
c = cards[counter]
if c[1] == suit
Yes
if (suit_ace_dict.get(c[0])) != None
Yes
total = total + *int*(suit_ace_dict[c[0]])
No
total = total + *int*(c[0])
No
counter = counter + 1
No
Return total

Flowchart: calculate_suit_range


calculate_suit_range(*deck*)
suit_ace_dict = {'J': "11", 'Q': "12", 'K': "13", 'A': "14"}
card1 = deck[0]
card2 = deck[1]
if suit_ace_dict.get(card1[0]) != None
Yes
if suit_ace_dict.get(deck[1][0]) != None
Yes
Return abs(*int*(suit_ace_dict[card1[0]]) - *int*(suit_ace_dict[card2[0]]))
No
Return abs(*int*(suit_ace_dict[card1[0]]) - *int*(card2[0]))
No
if suit_ace_dict.get(card2[0]) != None
Yes
Return abs(*int*(card1[0]) - *int*(suit_ace_dict[card2[0]]))
No
Return abs(*int*(deck[0][0]) - *int*(deck[1][0]))

Winner Test Cases



Test Cases	Description	Required Output	Actual Output
check_trick_winner(("K", "♥"), ("9", "♥"), "♠", "computer")	This is to check whether the player with the highest card from two cards with same suit where the suit is not trumps is chosen as winner.	"You won"	"You won"
check_trick_winner(("8", "♦"), ("9", "♦"), "♦", "player")	This is to check whether the player with the highest card from two cards with same suit where the suit is trumps is chosen as winner.	"Computer won"	"Computer won"
check_trick_winner(("10", "♣"), ("J", "♥"), "♣", "computer")	This is to check when the first player("player") plays a card in trump suit and the other player("computer") plays a card which is not in the trump suit. In this case the first player("player") is chosen as the winner.	"You won"	"You won"
check_trick_winner(("Q", "♦"), ("9", "♥"), "♥", "computer")	This is to check when the second player("computer") plays a card in trump suit and the first player("player") plays a card which is not in the trump suit. In this case the second player("computer") is chosen as the winner.	"Computer won"	"Computer won"
check_trick_winner(("Q", "♠"), ("A", "♦"), "♥", "player")	This is to check when both the players' cards do not have the same suit and neither of those suits are the trumps. In this case the one who leads the trick who is the player will win the trick (specified by the last parameter).	"You won"	"You won"
check_trick_winner(("10", "♦"), ("10", "♠"), "♣", "computer")	This is to check when both the players' cards do not have the same suit and neither of those suits are the trumps. In this case the one who leads the trick who is the computer will win the trick (specified by the last parameter).	"Computer won"	"Computer won"