1) Numbered steps that identify the key functionality of my program
    1. Get player name
    2. Ask whether game is single player or multiplayer – method with no input and string output
    3. Single player game method
        a. Make a new card deck – method for making new card deck
        b. Get a starting hand of 2 random cards for the player and dealer – this will be a method
        c. Print the dealer's first card – print card list method, specify only one card to be printed
        d. Do player's turn and get the score – play hand method
            i. Print player's turn
            ii. Print player's card list – this will be a method
            iii. Determine if the player has blackjack, and return if so
            iv. Go through hit or stand with the player and stop once they hit 21 or choose to stand – this will be a method
            v. Get the score – this will be a method
        e. Do dealer's turn and get the score – play hand method
            i. Print dealer's turn
            ii. Determine if the player has blackjack, and return if so
            iii. Go through hit or stand until the dealer has a score greater than 16 – this will be a method
            iv. Get the score – this will be a method
        f. Compare the scores and print the result, 2 scores at a time (player and dealer) – get score method given both of the scores
    4. Multiplayer game method
        a. Make a new card deck – method for making new card deck
        b. Get a starting hand of 2 random cards for the player, computer, and dealer – this will be a method
        c. Print the dealer's first card – print card list method, specify only one card to be printed
        d. Do player's turn and get the score – play hand method
            i. Print player's turn
            ii. Print player's card list – this will be a method
            iii. Determine if the player has blackjack, and return if so
            iv. Go through hit or stand with the player and stop once they hit 21 or choose to stand – this will be a method
            v. Get the score – this will be a method

e. Do computer's turn and get the score – play hand method
    i. Print dealer's turn – this will be a method
    ii. Determine if the player has blackjack, and return if so
    iii. Go through hit or stand until the computer has a score greater than or equal to 10 more than the dealer's first card. (i.e. – if dealer card is 3, computer will hit until its score is 13 or greater) – this will be a method
    iv. Get the score – this will be a method
f. Do dealer's turn and get the score – play hand method
    i. Print dealer's turn – this will be a method
    ii. Determine if the player has blackjack, and return if so
    iii. Go through hit or stand until the dealer has a score greater than 16 – this will be a method
    iv. Get the score – this will be a method
g. Compare the player and dealer score comparison and print the result – get score method given both of the scores
h. Compare the computer and dealer score comparison and print the result – get score method given both of the scores

2) Functions that my program will have

```
# Creates a full card deck
def newCardDeck():
  return cardDeck (dictionary value)

# prompts user if they want to hit or stand, and returns value
def hitOrStand():
  return userInput (string value)

# gets a random card and removes it from the deck, returns deck, and all values of the card
def getCard(cardDeck (dictionary)):
```

return returnDict (dictionary of card deck (dictionary), suite (string), card (int or string), and card value (int))

gives returns two cards as a list to a player

def startingHand(cardDeck (dictionary)):

return returnDict (dictionary of card deck (dictionary), and player card list (list of lists))

gets the score from a player's card list and prints and returns the value

def getScore(cardList (list), name (string)):

return score (int value)

Prints the card list of a player

def printCardList(cardList (list), numberOfCards (int), name (string)):

no return value

gets result of the game between two players, and a bet if it applies

def gameResult(playerScore (int), dealerScore (int), name (string)):

return winner (string)

Dealer logic is to hit if score is less than or equal to 16

def dealerLogic(dealerScore (int)):

return either “hit” or no value

hits only if player score is less than 10 over the dealer's first card

def myLogic(score (int), dealerFirstCardValue (int)):

return either “hit” or an empty string

```
# hits a card for a player, removes card from deck and returns deck and the card (card, value, and suite)

def hit(playerType (string), playerCardList (list), cardDeck (dictionary)):

    return returnDict (player card list (list), card deck (dictionary))


# Setup for playing a hand for a character

def playHand(playerType (string), dealerFirstCard (int), cardDeck (dictionary), playerCardList (list)):

    return score (int)


# Setup for single player game with player and dealer

def singlePlayerGame(name (string)):

    no return value


# Setup for multiplayer game with player, computer, and dealer

def multiPlayerGame(name (string)):

        no return value


# Gets single player or multiplayer input

def singleOrMultiPlayer():

    return “single player” or “multiplayer” based on user input


# Gets input for play again or not

def playAgain():

        returns true or false based on user input
```

3) Potential unit tests

   My first unit test would be testing singlePlayerGame/multiPlayerGame with an empty string inputted.

   The input would be singlePlayerGame(“”) to test what happened if I inputted a blank name. The expected output would be it constantly printing a blank name rather than an actual name, but the code would still run.

   My second unit test is one I actually used to check my code. I was testing the gameResult() method.

   The input would be gameResult(score1, score2, name), which the expected output would print who won and by what means, then return the winner name as a string. There were so many conditional statements that a unit test was used to check that all possibilities were functioning correctly.