3a. Provide a written response that does all of the the following:

Approx. 150 words (for all subparts of 3a combined)

i. Describes the overall purpose of the program

ii. Describes what functionality of the program is demonstrated in the video.

iii. Describes the input and output of the program demonstrated in the video

1. The overall purpose of the program is to allow the user to play Blackjack, with the goal of defeating the dealer by obtaining a hand of cards with a value as close to 21 as possible.
2. The function of the program is to generate a starting hand of two cards with values ranging from 1-10 for the dealer and the player, then allow the player to “hit”, or draw a card, or “stand”, which means their hand is compared against the dealer’s, through the use of buttons. Depending on whether the player wins or loses, a different message is displayed through a label beneath the cards.
3. An input demonstrated in the video is the “Restart” button. When this button is pressed, the label says that the game is restarting, and the current round’s hands are reset to new hands. The text in labels, and several other variables are also reset to their default/initial values in order to facilitate a new round being played.

3b. Capture and paste two program code segments you developed during the administration of this task that contain a list (or other collection type) being used to manage complexity in your program.

Approx. 200 words (for all subparts of 3b combined, exclusive of program code)

i. The first program code segment must show how data has been stored in the list.

ii. The second program code segment must show the data in the same list being used, such as creating new data from the existing data or accessing multiple elements in the list, as part of fulfilling the program's purpose.

```
let ranks = ["Ace": 1, "Two": 2, "Three": 3, "Four": 4, "Five": 5, "Six": 6, "Seven": 7, "Eight": 8, "Nine":
    9, "King": 10, "Queen": 10, "Jack": 10]
```

```
        for suit in suits{
            for rank in ranks{
                let newCard = Card(suit: suit, rank: rank.key, value: rank.value)
                blankDeck.append(newCard)
            }
        }
```

Then, provide a written response that does all three of the following

iii. Identifies the name of the list being used in this response

iv. Describes what the data contained in the list represents in your program.

v. Explains how the selected list manages complexity in your program code by explaining why your program code could not be written, or how it would be written differently, if you did not use the list.

1. The name of the list, which is a dictionary, is ranks.
2. The data contained in the list represents the numerical values of each card in a deck, from 1 to 10, when applied to the context of a blackjack hand.
3. This list manages complexity by assigning numerical values to every card, and allows them to be easily referenced and used. For example, when creating a new card, rank.value can be referenced easily to obtain the card's numerical value. The program code would be more complex than necessary if this dictionary was not used, as the numerical value of a card might have had to be assigned every time said card was drawn.

3c. Capture and paste two program code segments you developed during the administration of this task that contains a student-developed procedure that implements an algorithm used in your program and a call to that procedure.

Approx. 200 words (for all subparts of 3c combined, exclusive of program code).

i. The first program code segment must be a student-developed procedure that:

   a. Defines the procedure's name and return type (if necessary)

   b. Contains and uses one or more parameters that have an effect on the functionality of the procedure

   c. Implements an algorithm that includes sequencing, selection and iteration

ii. The second program code segment must show where your student-developed procedure is being called in your program.

iii. Describes in general what the identified procedure does and how it contributes to the overall functionality of the program.

iv. Explains in detailed steps how the algorithm implemented in the identified procedure works. Your explanation must be detailed enough for someone else to recreate it.

```
    //Code for the dealer's turn, when the player has finished drawing cards
    func dealerTurn(){

        if gameStage == 1{
            let randomIndex = Int.random(in: 0..<newDeck.count)
            let card = newDeck[randomIndex].image
            while dealerTotal < 17{
                    self.cardImage1.image = UIImage(named: card)
                    self.dealersHand.append(Int(card.suffix(2))!)
                    print((self.dealersHand))
                    self.dealerTotal = self.dealersHand.reduce(0, +)
                    print(self.dealerTotal)
                    self.infoUpdate()
                    self.dealerCount += 1
            }
            if dealerTotal >= 17{
                cardImage1.image = UIImage(named: card)
            }
        }
        if gameStage == 0{
            let randomIndex = Int.random(in: 0..<newDeck.count)
            let card = newDeck[randomIndex].image
                self.cardImage1.image = UIImage(named: card)
                self.dealersHand.append(Int(card.suffix(2))!)
                print((self.dealersHand))
                self.dealerTotal = self.dealersHand.reduce(0, +)
                print(self.dealerTotal)
                self.infoUpdate()
                self.dealerCount += 1

        }
    }
```

```
//Code for when the player stands (checks winner)
@IBAction func standButtonPressed(_ sender: Any) {

    dealerTurn()
```

The identified procedure is called when the “Stand” button is pressed, and contributes to the overall functionality of the program by first checking whether the “game stage” is 0, meaning it is the start of a new round, or 1, meaning it is the dealer’s turn after the player has finished hitting. If it is the start of a new round, a random card is generated, the card’s image is placed onto the image view, the card is added to the dealer’s hand, the total value of the hand is calculated, the information label under the cards is updated, and the dealer’s “count”, or number of cards drawn, increases by 1. If it is not the start of a new round, the code described for the start of a round runs, but inside a while loop which checks if the dealer’s hand is equal to 17 or higher yet. If the dealer’s hand is already 17 or higher, the code does not run and instead the image view “flips the card over” and displays the second card drawn by the dealer at the start of the round instead of the card back.

Steps of the algorithm:

1. Check if gameStage is equal to 0 (start of round) or 1 (after player finishes hitting and has stayed)
2. If it is 0,
3. Generate a random card
4. Change the image view's displayed image to the corresponding image of that card
5. Append the value of the card to the dealer's hand
6. Add the value of the card to the total value of the dealer's hand
7. Update the label beneath the cards
8. Increase the count of the number of cards drawn by the dealer by 1
9. If gameStage is 1,
10. Repeat steps 3 to 8, but inside a loop that only runs if the total value of the dealer's hand is less than 17
11. If the hand is already 17 or higher when gameStage is 1 (the dealer's starting hand of two cards has a value >= 17), flip the card over and display the second card's image instead of the card back.

3d. Provide a written response that does all three of the following:

Approx. 200 words (for all subparts of 3d combined, exclusive of program code).

i. Describes two calls to the procedure identified in written response 3c. Each call must pass a different argument(s) that causes a different segment of code in the algorithm to execute.

First Call:

Second Call:

ii. Describes what condition(s) is being tested by each call to the procedure.

Conditio n(s) tested by the first call:

Condition(s) tested by the second call:

iii. Identifies the result of each call.

Result of the first call:

Result of the second call:

1. The first call is when gameStage is equal to 0 (start of round), and the second call is when gameStage is equal to 1 (after the player finishes hitting and has stayed).
2. The first call generates a new card and adds it to the dealer's hand, as described in steps 3-8 of the procedure. The second call does the same thing until the total value of the dealer's hand is 17 or higher, but if the two starting cards of the dealer's hand already have a total value of 17 or higher, the procedure does not run and instead the

image view is changed to the image of the dealer's second card instead of the card back.

3. The result of the first call is that a new card is generated and added to the dealer's hand and the image of it is added to the image view as earlier described. The result of the second call depends on the value of the dealer's two starting cards. Let us assume that the dealer's starting hand has a value of 7. The while loop would keep running the card generation code while the hand's value is less than 17, so let us say that the first card generated has a value of 8, which means that the value is now 15, so the code runs again and now the dealer draws a card with a value of 4, and now has a hand of 19, so the code stops.