CS 8: Introduction to Computer Programming with Python
Summer 2017

Project 2

Assigned: Thursday, June 22 **Due:** Thursday, July 13 11:59 PM

Overview

Blackjack, or 21, is the most widely-played casino banking game in the world. The goal is to build a hand of cards that values as close to 21 as possible without exceeding it. First, here is the terminology used in the game.

- **Hit:** Take another card for your hand
- **Stand:** Stop taking cards
- **Bust:** Exceed a value of 21 in your hand
- **Push:** Tie with the dealer

The cards are valued as follows.

Rank	Value
Ace	11 or 1
2–10	Face value
Jack, Queen, King	10

Of particular note, an Ace can be considered either value 11 or 1. Given rational strategy, it will be considered 11 unless that will cause a bust, in which case it will be considered 1.

In each hand, play proceeds as follows.

1. The dealer takes one card, which the player can see.
2. The player is given two cards.
3. The player has the option to *hit*, or take one additional card. The player can hit as many times as desired, until they reach or exceed 21, or decide to *stand* (stop taking cards).

4. If the player did not *bust* (exceed 21), the dealer takes cards until she reaches 17 or higher (i.e., the dealer will hit on 16 or fewer points, and stand on 17 or more points).

5. If the player busts, she immediately loses, and the dealer will not draw more cards. If the dealer busts, the player wins. If the player and dealer have the same total, it is a push. Otherwise, the winner is the one with higher point total.

6. If the player loses, she loses her bet. If the player wins, she wins her bet. If the hand is a push, the bet is returned (i.e., no gain or loss).

In this project, you will program a functional Blackjack game by first coding the basic logic of the game, then gradually adding features.

Lab Session 1, Jun 22

Activity 1

In this activity, you will code the mainline logic of the game, and a function for the task of a single hand. The mainline logic will include managing the player's name and the amount of money in their account. The function for playing a hand will make some simplifications to the game, rather than tackling a full Blackjack game all at once (recall our design strategy of *reduction*). For instance, instead of selecting a card randomly from a full deck and then determining its value, you will instead randomly generate a card's value from between 2 and 11. You will also assume that the player always bets $25.

Note that randomly generating a card's value in this way is not a very accurate way to simulate drawing cards. For instance, in a real deck, there are more cards worth 10 (including all of the face cards) than there are cards worth 5. You can probably think of other ways in which this version of Blackjack is not exactly the same as real-life Blackjack. You will add features to make the game more realistic in later activities (*generalization*).

1. Write a `main()` function that asks the player for their name, and initializes their account to $1,000.

2. Add a loop to the `main()` function that will play a hand of blackjack, then ask the player if they want to play another hand—*unless* their account is out of money! If they go broke, you should quit the game.

   The task of playing a hand will be handled by the `play_hand(name)` function (which does not yet exist). This will be a value-returning function that returns the change in the player's money (positive if they win, negative if they lose), so you should use this value to update their amount after each hand.

3. Next, you will need to write the `play_hand(name)` function. This is the most complex part of the activity. You may need to sketch out your ideas in flowcharts, pseudocode, or as comments, before you start coding it for real.

   In `play_hand`, you will need to keep track of the dealer's points and player's points (that is, the value of both players' hands). Every time one of them receives a card, you will need to increase their points.

   First, deal one card to the dealer, then two to the player. If the player has a total of less than 21, ask the player whether they want to hit or stay. If they choose to hit, deal them another card. Continue to offer cards until the player chooses to stay, or until their total is 21 or greater. After each card is drawn, print the resulting state (see below).

   Once the player is done, it is the dealer's turn. If the player has bust (received a total over 21), the dealer automatically wins, so they won't need to take any cards. Otherwise, keep dealing cards to the dealer until they reach or exceed 17 (the dealer

will hit if their total is 16 or less). Again, after each card is given, print both players' totals.

Finally, you must determine the winner. If the player busts, then the dealer wins, and `play_hand` should return -25 (the player loses their $25). If the dealer busts, then the player wins, and the function should return 25 (the player wins $25). Otherwise, whoever has the higher total wins; if both have the same total, then the hand is a push (tie), and the function should return 0 (the bet is returned, so no change in the account).

Here is an example run after completing this stage:

```
Name? Bill
Bill has $1,000

Dealer received card of value 2
Bill received card of value 9
Bill received card of value 5
Dealer total: 2
Bill total: 14
Move? (hit/stay) h

Bill received card of value 6
Dealer total: 2
Bill total: 20
Move? (hit/stay) s


Dealer received card of value 4
Dealer total: 6
Bill total: 20

Dealer received card of value 5
Dealer total: 11
Bill total: 20

Dealer received card of value 10
Dealer total: 21
Bill total: 20
Dealer wins

Bill has $975

Play again? (y/n) y
Dealer received card of value 4
```

```
Bill received card of value 6
Bill received card of value 7
Dealer total: 4
Bill total: 13
Move? (hit/stay) h

Bill received card of value 10
Dealer total: 4
Bill total: 23

Bill bust

Bill has $950

Play again? (y/n) y
Dealer received card of value 3
Bill received card of value 11
Bill received card of value 10
Dealer total: 3
Bill total: 21

Dealer received card of value 7
Dealer total: 10
Bill total: 21

Dealer received card of value 3
Dealer total: 13
Bill total: 21

Dealer received card of value 11
Dealer total: 24
Bill total: 21
Dealer bust

Bill has $975

Play again? (y/n) y
Dealer received card of value 4
Bill received card of value 10
Bill received card of value 10
Dealer total: 4
Bill total: 20
Move? (hit/stay) s
```

```
Dealer received card of value 4
Dealer total: 8
Bill total: 20

Dealer received card of value 8
Dealer total: 16
Bill total: 20

Dealer received card of value 4
Dealer total: 20
Bill total: 20
Push

Bill has $975

Play again? (y/n) y
Dealer received card of value 2
Bill received card of value 5
Bill received card of value 10
Dealer total: 2
Bill total: 15
Move? (hit/stay) s


Dealer received card of value 11
Dealer total: 13
Bill total: 15

Dealer received card of value 5
Dealer total: 18
Bill total: 15
Dealer wins

Bill has $950

Play again? (y/n) y
Dealer received card of value 8
Bill received card of value 10
Bill received card of value 10
Dealer total: 8
Bill total: 20
Move? (hit/stay) s


Dealer received card of value 9
```

```
Dealer total: 17
Bill total: 20
Bill wins

Bill has $975

Play again? (y/n) n
```

Once you have completed these steps, show your TA your progress so that you get credit for completing the activity.

End Lab Session 1

Lab Session 2, Jun 29

Activity 2

In this activity, you will allow the player to change their bet before each hand.

1. Write a function `input_bet(bet, money)` that accepts two parameters. The `bet` parameter is the current bet; this is needed so the player can choose to keep their bet the same without re-typing it. The `money` parameter is the amount of money in the player's account; this is needed to ensure the player doesn't bet more than they have.

   In the `input_bet` function, ask the player how much they want to bet. You must use an input validation loop to make sure the player inputs a valid value.

   The bet must be a whole number, cannot be negative, and cannot be greater than the amount of money in the account. If the player enters an invalid value, you should print a message explaining the error. If the player presses enter without typing anything, this means they want to keep their existing bet value—be sure to check that they still have enough!

   An input of 0 is valid: it means the player wants to quit the game.

2. Next, modify `main()` to call `input_bet(bet, money)` before every hand **instead of** prompting for y/n for another hand. If the player bets 0, the program should quit. Otherwise, continue to play hands. Remember to pass the previous bet value to the `input_bet` function, so that if the player just presses enter, they keep the previous bet (and *not* a default bet!).

   Finally, add an argument to the `play_hand` function that represents the bet value. Modify this function so it returns the correct change in the player's account, instead of always winning or losing $25.

Here is an example run after completing this stage:

```
Name? Bill
Bill has $1,000
Bet? (0 to quit, Enter to stay at $25) 1500
You cannot bet more than $1,000
Bet? (0 to quit, Enter to stay at $25) Fifty
You must type a number
Bet? (0 to quit, Enter to stay at $25) 500
Dealer received card of value 10
Bill received card of value 10
Bill received card of value 10
Dealer total: 10
Bill total: 20
```

```
Move? (hit/stay) h

Bill received card of value 10
Dealer total: 10
Bill total: 30
Bill bust

Bill has $500

Bet? (0 to quit, Enter to stay at $500) 900
You cannot bet more than $500
Bet? (0 to quit, Enter to stay at $500)
Dealer received card of value 11
Bill received card of value 5
Bill received card of value 10
Dealer total: 11
Bill total: 15
Move? (hit/stay) h

Bill received card of value 11
Dealer total: 11
Bill total: 26
Bill bust

Bill has $0
Game over
```

Once you have completed these steps, show your TA your progress so that you get credit for completing the activity.

Activity 3

In this activity, you will save the player's game and allow them to resume. You will use file I/O as discussed in lecture. You will save and restore the player's name and the money in their account.

1. Write a function `save(name, money)` that saves a name and an amount of money to the plaintext file `blackjack.save`.

   Hint: In order to restore this information successfully, you should save these two pieces of information on separate lines of the file.

2. Write a function `restore()` that attempts to read the player's name and account total from the file `blackjack.save`. If it is successful, it should return the two values as a str and an int, respectively. If it fails, it should return `'', -1`. Since this is not a valid result, it indicates that the restore was unsuccessful (probably because there was no save file yet).

3. Modify function `main()` to call `save(name, money)` before quitting.

4. Modify function `main()` to call `restore()` when starting up. If the load was successful, ask the player if they would like to resume or start a new game. If they choose to start a new game, prompt them for their name, and re-initialize their account to $1000. If the load was unsuccessful, simply start a new game (ask for their name, reset account) without prompting the player to resume. You can test that this behavior works by deleting the `blackjack.save` file. Your program should work properly whether or not this file exists.

Here is an example run after completing this stage (nothing has changed yet, because there is no saved game to load):

```
Name? Bill
Bill has $1,000

Dealer received card of value 10
Bill received card of value 10
Bill received card of value 3
Dealer total: 10
Bill total: 13
Move? (hit/stay) h

Bill received card of value 4
Dealer total: 10
Bill total: 17
Move? (hit/stay) h
```

```
Bill received card of value 11
Dealer total: 10
Bill total: 28
Bill bust

Bill has $975

Bet? (0 to quit, Enter to stay at $25)
Dealer received card of value 10
Bill received card of value 10
Bill received card of value 6
Dealer total: 10
Bill total: 16
Move? (hit/stay) h

Bill received card of value 10
Dealer total: 10
Bill total: 26
Bill bust

Bill has $950

Bet? (0 to quit, Enter to stay at $25) 0
```

Here is an example of the load functionality when running the program again:

```
Resume saved game 'Bill'? (y/n) y
Bill has $950

Dealer received card of value 10
Bill received card of value 7
Bill received card of value 10
Dealer total: 10
Bill total: 17
Move? (hit/stay) s


Dealer received card of value 10
Dealer total: 20
Bill total: 17
Dealer wins

Bill has $925
```

```
Bet? (0 to quit, Enter to stay at $25) 0
```

Finally, here is an example of starting a new game even though a saved game exists:

```
Resume saved game 'Bill'? (y/n) n
Name? Phil
Phil has $1,000

Dealer received card of value 4
Phil received card of value 3
Phil received card of value 10
Dealer total: 4
Phil total: 13
Move? (hit/stay) h

Phil received card of value 10
Dealer total: 4
Phil total: 23
Phil bust

Phil has $975

Bet? (0 to quit, Enter to stay at $25) 0
```

Once you have completed these steps, show your TA your progress so that you get credit for completing the activity.

End Lab Session 2

Lab Session 3, Jul 06

Activity 4

In this activity, you will improve the card drawing functionality. In real life, you cannot draw 5 Aces in a single hand. However, if we're only generating random values, this is possible. As discussed before, our reduced version of Blackjack also makes it equally likely to get a card of value 5 and of value 10, but in reality there should be many more of value 10 than 5. You will fix this problem by using lists and tuples to represent a deck of cards to draw from. For each new hand, you will create a deck of cards, shuffle it, and draw from it as needed.

In this activity, you will represent a card as a tuple with 2 elements. The first element of the tuple will tell the card's rank, and the second element will tell the card's suit. You will store the rank using an int value, and the suit using a str value. Consider the rank encoding system below:

Rank	int encoding	Point value
Ace (A)	1	11
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8
9	9	9
10	10	10
Jack (J)	11	10
Queen (Q)	12	10
King (K)	13	10

For the suits, you will need escape characters, since card suits are not a character you can type on a standard keyboard. Use the following escape sequences to create strings with the suit characters:

Suit	Escape string
♠	`'\u2660'`
♡	`'\u2661'`
♢	`'\u2662'`
♣	`'\u2663'`

For example, a 5 of spades will be represented using the tuple `(5, '\u2660')`, and a queen of hearts will be represented as `(12, '\u2661')`. The deck will be represented as a list of 52 card tuples.

1. Write a function `new_deck()` that creates a new list, adds all 52 cards (one tuple for each rank/suit combination) to the list, and returns it.

   Hint: Use nested loops.

2. Write a function `shuffle_deck(deck)` that takes the deck as a parameter and shuffles it.

   Hint: Check the methods in the `random` module.

3. Write a function `value_of_card(card)` that takes a card tuple as a parameter and returns its point value according to the table above.

   Ex: `value_of_card((5, '\u2660'))` should return 5, since this encodes a 5 of spades, and `value_of_card((12, '\u2661'))` should return 10, since this encodes a queen of hearts.

4. Write a function `string_of_card(card)` that takes a card tuple as a parameter and returns that card as a string value for printing during the game.

   Ex: `string_of_card((5, '\u2660'))` should return `'5\u2660'`, which will print as 5♠, and `string_of_card((12, '\u2661'))` should return `'Q\u2661'`, which will print as Q♡.

5. Modify `play_hand()` to utilize this new functionality. At the start of a hand, create and shuffle a deck of cards. Instead of randomly generating cards, draw the "top" card from the shuffled deck. Be sure to remove it from the deck so you don't draw the same card over and over!

   When giving a card to the player or dealer, use `value_of_card(card)` to determine how much to add to their total points, and when printing what card they received, use `string_of_card(card)`.

Here is an example run after completing this stage:

```
Name? Bill
Bill has $1,000
Bet? (0 to quit, Enter to stay at $25) 100
Dealer received 10♢
Bill received 6♢
Bill received 2♠
Dealer total: 10
Bill total: 8
Move? (hit/stay) h

Bill received A♠
Dealer total: 10
Bill total: 19
Move? (hit/stay) s
```

```
Dealer received A♣
Dealer total: 21
Bill total: 19
Dealer wins

Bill has $900

Bet? (0 to quit, Enter to stay at $100) 0
```

Once you have completed these steps, show your TA your progress so that you get credit for completing the activity.

Activity 5

In this activity, you will modify your program to improve its output. Instead of keeping track of only the total points of the player and dealer, you will keep track of their full hand as a list of card tuples. This way, you can display the player's and dealer's full hands after each draw.

As an example, if the player draws a 5 of hearts, 3 of spades, and queen of diamonds, their hand would be represented as the following list of card tuples:
`[(5,'\u2661'), (3,'\u2660'), (12,'\u2662')]`

1. Write function `string_of_hand(hand)` that takes a list of card tuples as a parameter, and returns a string that represents this hand of cards. Use the `string_of_card(card)` function to determine the string for each card, then combine them into one string.

   Ex: The hand `[(5,'\u2661'), (3,'\u2660'), (12,'\u2662')]` should return the string that will print as 5♡ 3♠ Q♢

2. Write function `value_of_hand(hand)` that takes a list of card tuples as a parameter, and returns the value of the hand. Use the `value_of_card(card)` function to return the value for each card, then combine them into a single value.

   Ex: The hand `[(5,'\u2661'), (3,'\u2660'), (12,'\u2662')]` has value 18.

3. Modify the function `play_hand` so that it keeps track of the full hands of the player and dealer, rather than just the total points for their hand. Both of these hands should be encoded as lists of card tuples, as described above. Each time a player is given a card, remove it from the deck and add it to their hand. Modify your output so that it prints the player's and dealer's hands instead of only the individual cards as they are drawn.

4. Finally, modify your function `value_of_hand(hand)` so that it correctly adds the value of Ace. This is a challenge!

   Recall the rule: if an Ace of value 11 would make the hand bust, it is considered to be value 1 instead. Thus, an Ace alone is worth 11, an Ace with a King is worth 21, two Aces is worth 12, two Aces with a King is worth 12, and two Aces with two Kings is worth 22.

   Hint: One way to solve this is to create a list of values for each card. For example, if the player has two Aces, first create the list of values `[11, 11]`, then modify it according to the rule. This is not the only way! If you come up with something different, feel free to use it!

Here is an example run after completing this stage:

```
Name? Bill
Bill has $1,000
```

```
Bet? (0 to quit, Enter to stay at $25) 100

Bet: $100
Dealer's hand: 10♡
Value: 10
Bill's hand: J♠ 2♣
Value: 12

Move? (hit/stay) h

Bet: $100
Dealer's hand: 10♡
Value: 10
Bill's hand: J♠ 2♣ 3♣
Value: 15

Move? (hit/stay) h

Bet: $100
Dealer's hand: 10♡
Value: 10
Bill's hand: J♠ 2♣ 3♣ 7♠
Value: 22

Bill bust

Bill has $900

Bet? (0 to quit, Enter to stay at $100)

Bet: $100
Dealer's hand: 10♢
Value: 10
Bill's hand: 4♡ 2♣
Value: 6

Move? (hit/stay) h

Bet: $100
Dealer's hand: 10♢
Value: 10
Bill's hand: 4♡ 2♣ 3♣
Value: 9

Move? (hit/stay) h
```

```
Bet: $100
Dealer's hand: 10♢
Value: 10
Bill's hand: 4♡ 2♣ 3♣ 9♣
Value: 18

Move? (hit/stay) h

Bet: $100
Dealer's hand: 10♢
Value: 10
Bill's hand: 4♡ 2♣ 3♣ 9♣ A♡
Value: 19

Move? (hit/stay) s


Bet: $100
Dealer's hand: 10♢ 4♣
Value: 14
Bill's hand: 4♡ 2♣ 3♣ 9♣ A♡
Value: 19


Bet: $100
Dealer's hand: 10♢ 4♣ K♠
Value: 24
Bill's hand: 4♡ 2♣ 3♣ 9♣ A♡
Value: 19

Dealer bust

Bill has $1000

Bet? (0 to quit, Enter to stay at $100)

Bet: $100
Dealer's hand: A♣
Value: 11
Bill's hand: 10♣ 7♣
Value: 17

Move? (hit/stay) h
```

```
Bet: $100
Dealer's hand: A♣
Value: 11
Bill's hand: 10♣ 7♣ 7♢
Value: 24

Bill bust

Bill has $900

Bet? (0 to quit, Enter to stay at $100)

Bet: $100
Dealer's hand: 8♠
Value: 8
Bill's hand: 6♣ K♡
Value: 16

Move? (hit/stay) h

Bet: $100
Dealer's hand: 8♠
Value: 8
Bill's hand: 6♣ K♡ A♠
Value: 17

Move? (hit/stay) s


Bet: $100
Dealer's hand: 8♠ 5♠
Value: 13
Bill's hand: 6♣ K♡ A♠
Value: 17


Bet: $100
Dealer's hand: 8♠ 5♠ 8♢
Value: 21
Bill's hand: 6♣ K♡ A♠
Value: 17

Dealer wins

Bill has $800
```

```
Bet? (0 to quit, Enter to stay at $100)

Bet: $100
Dealer's hand: 4♢
Value: 4
Bill's hand: 2♢ J♡
Value: 12

Move? (hit/stay) s


Bet: $100
Dealer's hand: 4♢ A♡
Value: 15
Bill's hand: 2♢ J♡
Value: 12


Bet: $100
Dealer's hand: 4♢ A♡ A♢
Value: 16
Bill's hand: 2♢ J♡
Value: 12


Bet: $100
Dealer's hand: 4♢ A♡ A♢ 10♡
Value: 16
Bill's hand: 2♢ J♡
Value: 12


Bet: $100
Dealer's hand: 4♢ A♡ A♢ 10♡ 8♢
Value: 24
Bill's hand: 2♢ J♡
Value: 12

Dealer bust

Bill has $900

Bet? (0 to quit, Enter to stay at $100) 0
```

Once you have completed these steps, show your TA your progress so that you get credit for completing the activity.

Bonus Activity

This optional activity is less guided. Completing it will result in up to 8 bonus project points. In this activity, you will output the money the player has after each hand, in order to graph the result using a spreadsheet program.

A `csv` file, or comma-separated values, is a set of rows where each field is separated by a comma. In this bonus activity, after each hand, output the hand number and total money the player has, to a row in the file `blackjack.csv`. The data, including the hand number, should continue to accumulate even if the player has saved and resumed their game.

An example of such an output might be:

```
1,975
2,950
3,975
4,1000
5,1025
6,1050
7,1025
...
```

Once you accomplish this, play at least 20 hands, then open your `blackjack.csv` file in a spreadsheet program such as Excel or Google Sheets. Draw a line graph showing the amount of money you had over those hands.

You do not need to show your TA the result of completing this activity. Instead, simply submit your code with this modification, **and** upload an image of the graph that you created.

End Lab Session 3