Murach's Python Programming

Case Study by Chapter: Blackjack

For this case study, you'll use the programming skills that you learn in *Murach's Python Programming* to develop a program for a card game called Blackjack. By the time it's finished, this program will let the user play Blackjack against a computer dealer.

After you read chapter 2, you can develop a first version of the program that calculates the amount that the user wins or loses for each hand. Then, after you complete most of the other chapters, you can enhance and improve this program. By chapter 16, you'll create an object-oriented version of this program that includes betting and keeps track of the amount of money that a player has won or lost. And after chapters 17 and 18, you can add the use of a database and a GUI.

General guidelines

Naming

- When creating the folder and file names for your programs, please use the conventions specified by your instructor. Otherwise, for a program that consists of a single file, use this naming convention: *first_last*_blackjack_ch*XX*.py where *first_last* specifies your first and last name and ch*XX* specifies the chapter number, as in ch05. For programs that have multiple files, store the files in a folder named *first_last*_blackjack_ch*XX*.
- When creating names for variables and functions, please use the guidelines and recommendations specified by your instructor. Otherwise, use the guidelines and recommendations specified in *Murach's Python Programming*.

User interfaces

- You should think of the user interfaces that are shown for the case studies as starting points. If you can improve on them, especially to make them more user-friendly, by all means do so.

Specifications

- You should think of the specifications that are given for the case studies as starting points. If you have the time to enhance the programs by improving on the starting specifications, by all means do so.

Top-down development

- Always start by developing a working version of the program. That way, you'll have something to show for your efforts if you run out of time. Then, you can build out that version of the program until it satisfies all of the specifications.
- From chapter 5 on, you should use top-down coding and testing as you develop your programs. You might also want to sketch out a hierarchy chart for each program as a guide to your top-down coding.

Chapter 2: Calculate the player's money

Create a program that calculates the amount of money that a player will have if the player gets a Blackjack, wins, loses, or pushes (ties).

Console

```
BLACKJACK!
Blackjack payout is 3:2

Starting money:   100
Bet amount:       5

ENDING MONEY FOR A...
Blackjack:        107.5
Win:              105.0
Push:             100.0
Lose:             95.0

Bye!
```

Specifications

- The program should accept integer or float entries.
- Assume the user will enter valid data.
- Getting a Blackjack pays out 3:2, which is 1.5 times the bet amount.
- The program should round the Blackjack payout to a maximum of two decimal places.

Chapter 3: Track the player's money

Create a program that keeps track of the money until the user exits the program.

Console

```
BLACKJACK!
Blackjack payout is 3:2
Enter 'x' for bet to exit

Starting money:  100

Bet amount: 5
Blackjack, win, push, or lose? (b/w/p/l): b
Money: 107.5

Bet amount: 5
Blackjack, win, push, or lose? (b/w/p/l): w
Money: 112.5

Bet amount: 10
Blackjack, win, push, or lose? (b/w/p/l): p
Money: 112.5

Bet amount: 10
Blackjack, win, push, or lose? (b/w/p/l): l
Money: 102.5

Bet amount: x
Bye!
```

Specifications

- Assume the user will enter valid data.

Chapter 4: Use functions to organize the program

Create a program that provides the same functionality as chapter 3, but use functions to organize the code for the program so it's easier to reuse and maintain.

Console

```
BLACKJACK!
Blackjack payout is 3:2
Enter 'x' for bet to exit

Starting money:   100

Bet amount: 5
You won.
Money: 105.0

Bet amount: 5
You lost.
Money: 100.0

Bet amount: 10
You lost.
Money: 90.0

Bet amount: 10
You lost.
Money: 80.0

Bet amount: 20
You pushed.
Money: 80.0

Bet amount: 20
You got a blackjack!
Money: 110.0

Bet amount: x
Bye!
```

Specifications

- Assume the user will enter valid data.
- Use the random module to determine whether the player gets a Blackjack, wins, pushes, or loses using the following odds:

```
Blackjack     5%
Win          37%
Push          9%
Loss         49%
```

Chapter 5: Test and debug the program

Specifications

- Create a list of valid entries and the correct results for each set of entries. Then, make sure that the results are correct when you test with these entries.
- Create a list of invalid entries. This list should include entries that test the limits of the allowable values. Then, handle the invalid integers (such as negative integers) in this program:

  The maximum amount of starting money should be 10,000.

  The minimum bet should be 5.

  The maximum bet should be 1,000.

  The bet can't be greater than the player's current amount of money.
- Don't handle other invalid entries that cause exceptions (such as the user entering a string like "x" for a monetary amount). You can do that after you read chapter 8.

Chapter 6: Use lists to store cards

Modify the program so it allows the player to play Blackjack against a computer dealer. Then, the program can update the player's money depending on the result of each round.

Console

```
BLACKJACK!
Blackjack payout is 3:2

Starting money: 100
Bet amount: 10

DEALER'S SHOW CARD:
6 of Hearts

YOUR CARDS:
2 of Clubs
7 of Hearts

Hit or stand? (hit/stand): hit

YOUR CARDS:
2 of Clubs
7 of Hearts
Jack of Spades

Hit or stand? (hit/stand): stand

DEALER'S CARDS:
6 of Hearts
4 of Hearts
7 of Spades

YOUR POINTS:     19
DEALER'S POINTS: 17

Hooray! You win!
Money: 110.0

Play again? (y/n): y

Bet amount: 5
...
```

Specifications

- Use a list to store the suit, rank, and point value for each card.
- Use a list of lists to store the cards in the deck. You can use two nested loops to create the deck of cards.
- Use a list of lists to store the dealer's hand and the player's hand.
- If necessary, learn the rules of Blackjack by researching it on the web.
- Use a standard 52-card deck of playing cards.
- The dealer must continue taking cards until the dealer has at least 17 points.
- Don't allow a player to "split" a hand or "double down."

Chapter 7: Use a file to store player's money

Update the program so it remembers the player's money even after the player ends the program.

Console

```
BLACKJACK!
Blackjack payout is 3:2

Money: 170.0
Bet amount: 170

DEALER'S SHOW CARD:
9 of Clubs

YOUR CARDS:
7 of Hearts
King of Diamonds

Hit or stand? (hit/stand): hit

YOUR CARDS:
7 of Hearts
King of Diamonds
5 of Diamonds

DEALER'S CARDS:
9 of Clubs
5 of Spades
7 of Clubs

YOUR POINTS:     22
DEALER'S POINTS: 21

Sorry. You busted. You lose.
Money: 0.0

Play again? (y/n): y

You are out of money.
Would you like to buy more chips? (y/n): y
Amount: 200
Money: 200.0
Bet amount:
```

Specifications

- Use a plain text file to store the money amount.
- Store the functions for writing and reading the money amount in a separate module than the rest of the program.
- When the program starts, it should read the player's money amount from a file.
- The program should write the money amount to a file any time the data is changed.
- If the money amount drops below the minimum bet (5), the program should give the player the option to buy more chips.

Chapter 8: Handle exceptions

Thoroughly test and update the program so it handles all exceptions that you encounter during testing.

Console

```
BLACKJACK!
Blackjack payout is 3:2

Could not read money from file.
Restarting with 100.

Money: 100.0
Bet amount: x
Invalid amount. Try again.
Bet amount: 50

DEALER'S SHOW CARD:
4 of Spades

YOUR CARDS:
Ace of Clubs
3 of Diamonds

Hit or stand? (hit/stand): hit

YOUR CARDS:
Ace of Clubs
3 of Diamonds
5 of Spades

Hit or stand? (hit/stand): stand

DEALER'S CARDS:
4 of Spades
7 of Diamonds
Queen of Clubs

YOUR POINTS:     19
DEALER'S POINTS: 21

Sorry. You lose.
Money: 50.0

Play again? (y/n): n

Come back soon!
Bye!
```

Specifications

- Handle the exception that occurs if the program can't find the data file.
- Handle the exceptions that occur if the user enters a string where an integer is expected.

Chapter 9: Improve number formatting

Update the program to improve the formatting of the numbers. In particular, apply the monetary format that's appropriate for your country to the money amount.

Console

```
BLACKJACK!
Blackjack payout is 3:2

Money: $95.00
Bet amount: 5

DEALER'S SHOW CARD:
7 of Spades

YOUR CARDS:
2 of Hearts
King of Clubs

Hit or stand? (hit/stand): hit

YOUR CARDS:
2 of Hearts
King of Clubs
8 of Spades

Hit or stand? (hit/stand): stand

DEALER'S CARDS:
7 of Spades
Queen of Clubs

YOUR POINTS:     20
DEALER'S POINTS: 17

Hooray! You win!
Money: $100.00

Play again? (y/n): n

Come back soon!
Bye!
```

Specifications

- Use the locale module to display the amount of money using formatting that's appropriate for your current locale.
- Make sure the program doesn't yield incorrect money amounts due to floating-point errors.

Chapter 11: Store the time for each session

Update the program so it begins by displaying the starting time of the session and ends by displaying the stop time and elapsed time of the session.

Console (top)

```
BLACKJACK!
Blackjack payout is 3:2
Start time: 11:29:52 AM

Money: $140.00
Bet amount:

...
```

Console (bottom)

```
...

Play again? (y/n): n

Stop time: 11:30:25 AM
Elapsed time: 00:00:32
Come back soon!
Bye!
```

Specifications

- When the program starts, display the start time at the top of the console using the 12-hour format and include hours, minutes, and seconds.
- When the program ends, display the end time using the 12-hour format as well as the elapsed time for the session.

Chapter 12: Use a dictionary to store card data

Update the program so it uses a dictionary to store the data for each card (rank, suit, and points). This shouldn't change the functionality of the program, but it should improve the readability of the code.

Console

```
BLACKJACK!
Blackjack payout is 3:2
Start time: 11:43:03 AM

Money: $110.00
Bet amount: 10

DEALER'S SHOW CARD:
6 of Clubs

YOUR CARDS:
9 of Clubs
Jack of Clubs

Hit or stand? (hit/stand): stand

DEALER'S CARDS:
6 of Clubs
Queen of Hearts
Queen of Diamonds

YOUR POINTS:     19
DEALER'S POINTS: 26

Yay! The dealer busted. You win!
Money: $120.00

Play again? (y/n): n

Stop time: 11:43:20 AM
Elapsed time: 00:00:16
Come back soon!
Bye!
```

Chapter 14: Use an object-oriented approach

Convert the program from procedural to object-oriented. This shouldn't change the functionality of the code, but it should make the code more modular, reusable, and easier to maintain.

Console

```
BLACKJACK!
Blackjack payout is 3:2
Start time: 11:43:03 AM

Money: $110.00
Bet amount: 10

DEALER'S SHOW CARD:
6 of Clubs

YOUR CARDS:
9 of Clubs
Jack of Clubs

Hit or stand? (hit/stand): stand

DEALER'S CARDS:
6 of Clubs
Queen of Hearts
Queen of Diamonds

YOUR POINTS:     19
DEALER'S POINTS: 26

Yay! The dealer busted. You win!
Money: $120.00

Play again? (y/n): n

Stop time: 11:43:20 AM
Elapsed time: 00:00:16
Come back soon!
Bye!
```

Specifications

- Use a Card class that provides attributes for storing the rank, suit, and points for a card. This class should also provide a method that returns a string that includes the rank and suit of the card.
- Use a Deck class that provides for a standard 52-card playing deck. This class should include methods that allow you to shuffle the deck and to deal cards from it.
- Use a Hand class to store the dealer's hand and the player's hand. This class should include methods that allow you to add a card, get a card, count the number of cards, and get the total points for a hand of Blackjack.

Chapter 15: Improve the Card and Hand objects

Use inheritance to improve your Card and Hand objects. Then, modify your program to take advantage of these improvements. This shouldn't change the functionality of the code, but it should make the code shorter and easier to read.

Console

```
BLACKJACK!
Blackjack payout is 3:2
Start time: 11:43:03 AM

Money: $110.00
Bet amount: 10

DEALER'S SHOW CARD:
6 of Clubs

YOUR CARDS:
9 of Clubs
Jack of Clubs

Hit or stand? (hit/stand): stand

DEALER'S CARDS:
6 of Clubs
Queen of Hearts
Queen of Diamonds

YOUR POINTS:     19
DEALER'S POINTS: 26

Yay! The dealer busted. You win!
Money: $120.00

Play again? (y/n): n

Stop time: 11:43:20 AM
Elapsed time: 00:00:16
Come back soon!
Bye!
```

Specifications

- Add a __str__ method to the Card class and use it in the program.
- Add an iterator to the Hand class and use it in your program.
- Modify the code in the program so it uses these new features.

Chapter 16: Implement the three-tier architecture

If the program doesn't already implement the three-tier architecture, implement it now. This shouldn't change the functionality of the code, but it should make the code more modular, reusable, and easier to maintain.

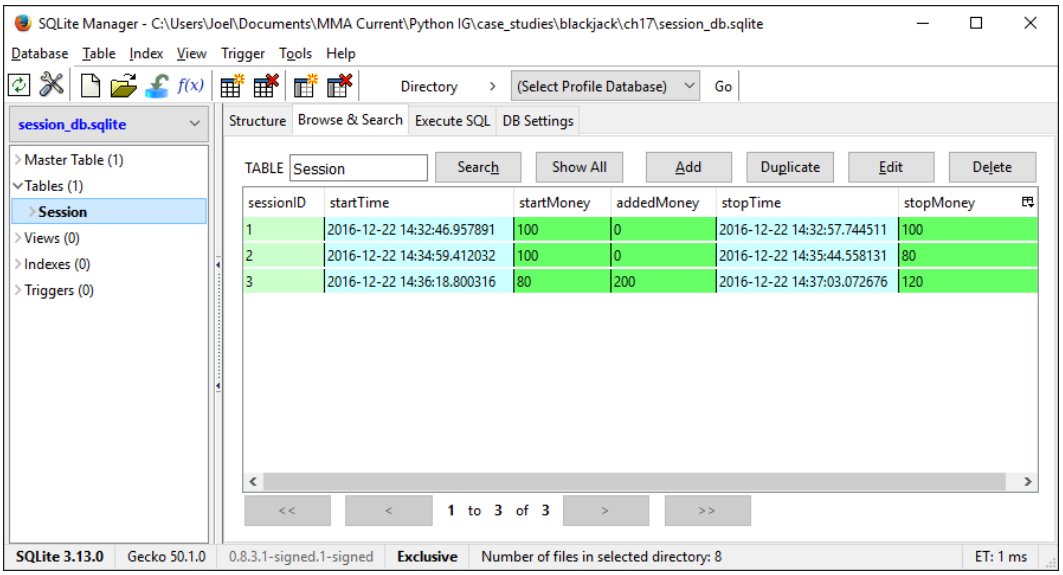
Specifications

- Use a file named ui to store the code for the user interface.
- Use a file named objects to store the code for the Card, Deck, and Hand classes.
- Use a file named db to store the functions that work with the data file.

Chapter 17: Use a database

Use a database to store session data for the program.

SQLite Manager with the Session table displayed



Specifications

- Use SQLite Manager to create a new database for the program.
- Use a single table named Session to store the data for a session of Blackjack playing. Here's a SQL statement that defines the columns and their data types for this table:

```
CREATE TABLE Session (
    sessionID   INTEGER PRIMARY KEY     NOT NULL,
    startTime   TEXT                    NOT NULL,
    startMoney  REAL                    NOT NULL,
    addedMoney  REAL                    NOT NULL,
    stopTime    TEXT                    NOT NULL,
    stopMoney   REAL                    NOT NULL
);
```

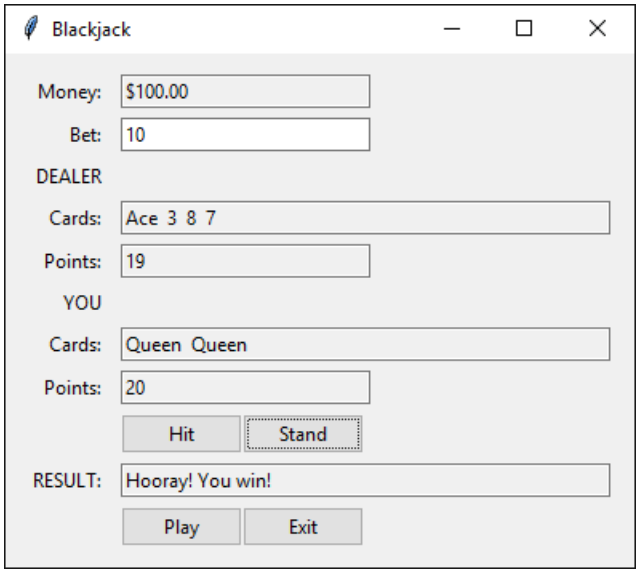
  To create this table, use SQLite's Execute SQL tab to run this SQL statement.

- Modify the objects module so it includes a Session class that stores all of the data for the session.
- Modify the db module so it provides all functions necessary to read and write session data.
- When the program starts, it should read the stop money amount from the most recent session. When the program stops, it should write the data for the session to the database.

Chapter 18: Use a GUI

Use a GUI to allow the user to play Blackjack.

GUI



Specifications

- The player must enter a bet in the Bet text field to be able to play.
- The player must click the Play button to start a new game.
- The player can click on the Hit and Stand buttons to hit or stand.
- The program should use the Money field to display the amount of money that the player has.
- The program should use the Bet field to get the amount of the bet from the player.
- The program should use the Cards and Points fields to display the cards and points for the dealer and the player. In the Cards fields, save horizontal space by not displaying the suit of the card.
- The program should use the Result field to display messages to the player. These messages should include whether the user needs to place a bet, whether the user has won or lost, and so on.
- Don't implement the feature that allows the player to add more money. If necessary, you can use the console version of the program to do that.