Murach's Python Programming

Case Study by Section: Blackjack

For this case study, you will develop a program for a Blackjack card game using the skills that you learned in *Murach's Python Programming*. This program lets the user play Blackjack against a computer dealer.

After you read section 1, you can develop a simple version of the Blackjack program. After you read section 2, you can add to this program and improve it. After you read section 3, you can create an object-oriented version of this program. And after you read section 4, you can create a version that uses a database and a GUI.

General guidelines

Naming

- When naming the folders for your programs, please use the conventions specified by your instructor. Otherwise, store the files for a program in a folder named *first_last_blackjack*_sec*X* where *first_last* specifies your first and last name and *X* specifies the section number, as in sec1.
- 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 for a case study. That way, you'll have something to show for your efforts if you run out of time. Then, you can build out that starting version of the program until it satisfies all of the specifications.
- In particular, you should use top-down coding and testing as you develop your programs as described in chapter 5. You might also want to sketch out a hierarchy chart for each program as a guide to your top-down coding.

Section 1: Create the program

Create a program that allows the user to play Blackjack against a computer dealer.

Console

```
BLACKJACK!
Blackjack payout is 3:2

Money: 100.0
Bet amount: 10

DEALER'S SHOW CARD:
9 of Diamonds

YOUR CARDS:
5 of Hearts
5 of Clubs

Hit or stand? (hit/stand): hit

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

Hit or stand? (hit/stand): stand

DEALER'S CARDS:
9 of Diamonds
Jack of Hearts

YOUR POINTS:     17
DEALER'S POINTS: 19

Sorry. You lose.
Money: 90.0

Play again? (y/n): n

Come back soon!
Bye!
```

Specifications

- If necessary, learn the rules of Blackjack by researching it on the web. For this program:
  - 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.”
- The program should accept integer or float entries for the bet amount.
- 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.

Section 1: Create the program (continued)

Specifications (continued)

- 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.
- When the program starts, it should read the player's money amount from a CSV file named money.txt.
- The program should write the player's money amount to a file any time the data is changed
- Store the functions for writing and reading the money amount in a separate module named db.py.
- 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 or float value is expected.
- The program should validate the bet amount.
    - The minimum bet should be 5.
    - The maximum bet should be 1,000.
    - The bet can't be bigger than the player's current amount of money.
- If the money amount drops below the minimum bet (5), the program should give the player the option to buy more chips.

Section 2: Improve the program

Use the skills you learned in section 2 to improve this program. This should improve the appearance of the console and 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!
```

Specifications

- Use the locale module to display the amount of money in the format for your current locale.
- Make sure the program doesn't yield incorrect money amounts due to floating-point errors.
- 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 amount of time that elapsed while the program was running.
- Use 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 make the code easier to read and understand.

Section 3: Create an object-oriented program

Convert the Blackjack 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!
```

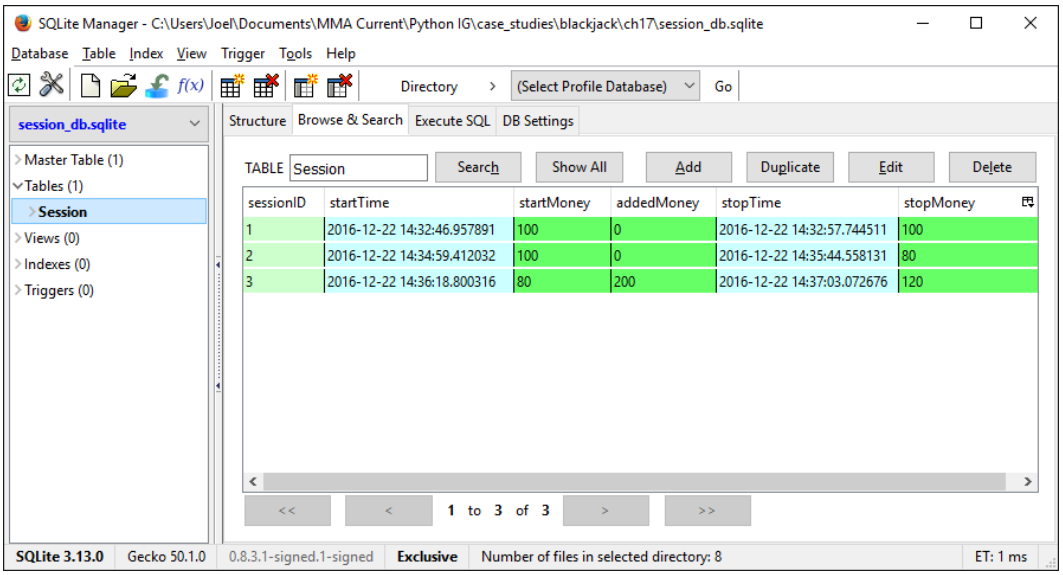
Specifications

- Use a Card class that provides attributes that store the rank, suit, and points for a card. This class should provide a __str__ 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 Blackjack hand. It should also include an iterator.
- 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.

Section 4: Use a database and a GUI

Use a database to store session data for the Blackjack program and a GUI to play it.

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 playing Blackjack. Here's a SQL statement that defines the columns and their data types for this table:

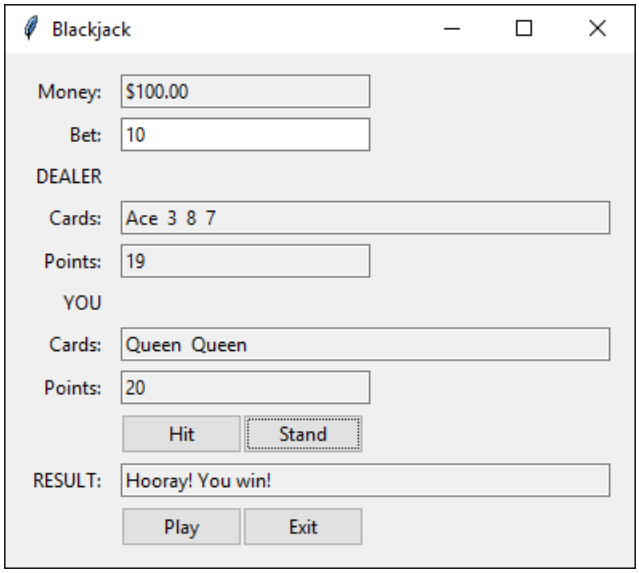
```
CREATE TABLE Session (
    sessionID   INTEGER PRIMARY KEY     NOT NULL,
    startTime   TEXT                    NOT NULL,
    startMoney  REAL                    NOT NULL,
    addedMoney  REAL                    NOT NULL,
    stopTime    TEXT                    NOT NULL,
    stopMoney   REAL   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

Section 4: Use a database and a GUI (continued)

GUI



Specifications

- The player must enter a bet in the Bet text field before starting to play.
- The player clicks the Play button to start a new game.
- The player clicks the Hit or Stand button to hit or stand.
- The program displays the player's money amount in the Money field.
- 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, you can 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.