Assignment P5

Due Date: June 18

Purpose

Thank goodness the last project is over! You now have sufficient Python knowledge that you can finally play a real *game*! To do this, you will write several functions that use parameters and return values. You will also have to bone up on your card playing skills.

Problem

The card game of Blackjack is addictive. Trouble is, if you really play it at a casino, you find your money leaving your wallet very quickly. Instead, why not play *virtual* Blackjack? That way, you will only lose your *virtual* money. With this in mind, *ESBN - Entertainment and Sports Betting Network* has hired you to write just such a game.

Basic Rules of Blackjack

The idea of the game is to finish a hand with more points than the dealer, also called the house. Numeric cards are worth the points shown (2 - 10); face cards, that is, king, queen, and jack, are each worth 10 points. The ace is worth 11 or 1, depending on which is more advantageous.

One hand of the game begins by the player placing a bet. The house then deals two cards face up to the player and two cards, only one of which is face up, to himself. The player then must decide if the current hand (the sum of the points) will beat the dealer's. This is tricky because one of the dealer's cards is covered. If the player thinks she can do better, then she "hits," or receives another card. The player can hit as often as she likes until she "stands." If at any point the player's points total more than 21, the game is over. If the player stands, then the dealer's card is revealed, and the dealer tries to surpass the player's point total by hitting, repeatedly if necessary. If the dealer goes over 21, then the player wins and gets double the bet back. Conversely, if the player loses, then the bet is lost.

It's not quite that easy...

There are many more nuances to the game; information can be found at `www.blackjackinfo.com` or other online sites. On that site under the **Practice** menu option, go to the **Online Blackjack Strategy Trainer** and click on the **Blackjack Strategy Trainer v2** to play on your computer (for free). This will give you an idea of how to play and set up your game.

Modified blackjack rules for this project

To make this project a little easier, you will implement some modified blackjack rules:

- There are an infinite number of decks for this game. This means you do not have to keep track of how many cards have been played.
- Dealers stands on **exactly** 17. This means that even if you have more, the dealer will not hit, and you win. Note the dealer can go **over** 17; it is only at **exactly** 17 that the dealer stands.
- Aces are **always** worth 11.
- Bets for each hand must be at least \$10, with increments of \$10. The player's bankroll should be immediately decremented accordingly.

- The game ends when either the player has less than \$10 or places a bet of \$0. In other words, the program should continue to loop until the user has lost all his money or inputs 0 for a bet.
- Both the player and the dealer can hit a maximum of **two** times. This rule makes it easier for you to create the right number of variables.
- No doubling, splitting, or insurance.
- Winning:
  - If at any point the player goes above 21, she loses that hand (and the bet).
  - If at any point the dealer goes above 21, the player wins the bet (2× the original bet).
  - If the dealer reaches 17 and the player has more, the player wins the bet as above.
  - If both the dealer and the player end with the same amount of points, that is referred to as a "push." The player gets the original bet back.
  - If the player is dealt 21 (blackjack!), and the dealer does not also have blackjack, then the player wins 2.5× the original bet.

Input

Before the game begins, the player will input the starting bankroll. Play then begins; before the cards are dealt, that is, before each new hand, the player must input the bet, which must be at least \$10 but can be larger, as long as the bet is evenly divisible by \$10. You may assume the player will input the values correctly; no error checking is necessary. A bet of \$0 signals the player quits and runs away with the winnings.

During each hand, the player will type either 'h' (for "hit") or 's' ("stand"). In the case of 'h', the player can hit up to two times until she stands or until she loses. When the hand is over, the player types in the next bet.

Output

- The current bankroll should be prominently displayed. After the cards are dealt, they should be displayed clearly, following the chart below:

Card	Output	Suit	Output	Python string
Jack	J	Hearts	♥	`"\u2665"`
Queen	Q	Diamonds	♦	`"\u2666"`
King	K	Clubs	♣	`"\u2663"`
Ace	A	Spades	♠	`"\u2660"`

  Thus, a hand consisting of the jack of diamonds and the 3 of clubs would be: J♦ 3♣.

- The dealer's hidden card should be shown with XX.
- As the game progresses, the points for each player should be shown.

- When the player stands, the dealer's hand should be displayed.
- At the end of each hand, the program should display whether the player won or lost and show the updated bankroll.

Sample Game

```
Bankroll:  $970.00
Bet?  10

Dealer:  4♠ XX -- 4 points
Player:  A♣ 2♠ -- 13 points
Hit/Stand?  h

Dealer:  4♠ XX -- 4 points
Player:  A♣ 2♠ 5♦ -- 18 points
Hit/Stand?  s

Dealer:  4♠ J♥ 6♥ -- 20 points
Player:  A♣ 2♠ 5♦ -- 18 points
Player loses.

Bankroll:  $960.00
Bet?  10

Dealer:  7♦ XX -- 7 points
Player:  Q♣ A♦ -- 21 points
BLACKJACK! Player wins!

Bankroll:  $975.00
Bet?  0

Thanks for playing!
```

Specifics

- You must design the program so that `main()` is only a "manager;" that is, it should call functions to do all the work. To this end, **no** input or output functions (`input()` or `print()`) are allowed in `main()`.
- You must have a function called `getCard()` that gets the next card and returns the value of the card as a numeric integer code. To this end, use the `random()` function to generate a random number between 2 and 14. 2 represents the card 2, 3 represents a 3, ..., 10 represents a 10, 11 represents a Jack, 12 represents a Queen, 13 represents a King, and 14 represents an Ace. Note that these numbers to not reflect the card's point value.
- You must have a function called `getSuit()` that randomly chooses a suit for the next card, and returns the string to display. This function should have no parameters. Note that this function works together with `getCard()` above to get all of the information for the next card.

- Other function(s) are up to you. Be sure to use return types and/or parameters correctly. This means using good programming practice, not just "making it work" in Python.
- All functions should have a descriptive comment as shown in class.
- If you find yourself writing the same (or essentially the same) code repeatedly, that indicates you need another function which you can then call repeatedly, using different arguments if need be.
- Do **not** use any global variables.

Notes

Once again, I urge you to start small and work your way to the final program. For example, you might start by writing a very short `main()` and then adding in one function at a time. Be certain that that function works before going on to the next.

While you are coding and debugging, use the `print()` function liberally so that you know that your values are correct up to that point. You can always delete extra statements later.

As usual, send me your source code by the 11:59:59 pm. The name of the file should be *last_name*P5.py as in `gousieP5.py`.

The answer is either `m` *or something else.*
– Professor Eva Ma (a grad school professor of mine)