Week 3 Assignment: Blackjack Strategy Calculations

Overview

In this assignment, you develop code to simulate the basic Blackjack strategies around "hitting" (getting another card) and "standing" (not getting another card). Specifically, in this assignment you will calculate the probability of going "bust" with different play strategies.

Some important aspects of Blackjack are quoted below, but as background, please read the Wikipedia page on Blackjack:

Blackjack is played one on one against the dealer. From Wikipedia:

> *The hand with the highest total wins as long as it doesn't exceed 21; a hand with a higher total than 21 is said to bust. Cards 2 through 10 are worth their face value, and face cards (jack, queen, king) are all worth 10. An ace's value is 11 unless this would cause the player to bust, in which case it is worth 1. A hand in which an ace's value is counted as 11 is called a soft hand, because it cannot be busted if the player draws another card.*

Initially, players are dealt two cards. They then must decide whether to keep receiving more cards. The terminology used here is:

- **Hit:** take another card
- **Stand:** take no more cards, let their current score stand

As the accumulated score of a hand increases, so does the probability of going "bust." When played in a casino, dealers are required to play according to a strict set of rules. In particular, they must keep dealing themselves new cards as long as their total value is below a certain threshold. Once that threshold has been reached, they MUST NOT deal themselves any more cards. Casinos set this threshold to maximize their chances of winning. According to Wikipedia:

> *There are two slightly different dealer strategies. In the "S17" game, dealer stands on all 17s. In the "H17" game, dealer hits on soft 17s; of course, he stands on hard 17s. (In either case, the dealer has no choice; he must or must not hit.) The H17 game is substantially less favorable to the player. Which game is customary depends on locality. Las Vegas Strip rules are about equally split.*

Please follow the steps outlined below.

Preparation

Create a new Python script called **blackjack.py** and set it up as a module. Initialize it with the following code:

```
import random

def get_card():
    return random.randint(1, 13)
```

This function will return a random value between 1 and 13 (ends included). We are modeling a card as an integer value between 1 and 13 (1 = ace, 2–10 number cards, {11,12,13} are jack, queen, king). We do not represent the suit since it does not matter in the game of Blackjack.

Note also that we are not representing a deck of cards (i.e., a collection of 52 cards with four of each rank). The code above essentially assumes an infinite deck of cards, which will change the calculated probabilities slightly compared to what you might find online.

Finally, create a test suite: **test_blackjack.py** and set it up as a **unittest** test case.

Step 1: Score Function

In **blackjack.py**, implement a function called "score":

```
def score(cards):
     ## TODO: fill this in
     return (total, soft_ace_count)
```

The argument to this function is a list of integers representing the cards in a Blackjack hand. This function returns a tuple. The first element of the tuple is the **total value** of the hand according to the scoring rules of Blackjack (see above; this is not simply the sum of the integers in the cards list). The second element of the tuple is the number of "soft" aces that remain in the hand after doing any conversions from 11 → 1 to keep the hand from going bust.

Some examples of card lists on the left and the corresponding tuple values on the right:

[3, 12] → (13, 0)
[5, 5, 10] → (20, 0)
[11, 10, 1] → (21, 0)
[1, 5] → (16, 1)
[1, 1, 5] → (17, 1)
[1, 1, 1, 7] → (20, 1)
[7, 8, 10] → (25, 0)

Add test cases to **test_blackjack.py** that correspond to the above examples and any other test cases that validate your implementation.

Step 2: Stand Function

In **blackjack.py**, implement a function called "stand":

```
def stand(stand_on_value, stand_on_soft, cards):
     total, soft_ace_count = score(cards)
     ## TODO: fill this in
     return True or False
```

The argument to this function is the stand-on value (e.g., 17), a Boolean value indicating whether the player will stand on a "soft" hand or just on a "hard" hand, and a list of integers representing the cards in a Blackjack hand. This function returns True if the player would "stand" on the hand, given the policy, and False otherwise. As you can see, the first thing this function does is call the score function. Fill in the rest of "stand" so that it returns the correct Boolean value.

Add test cases to **test_blackjack.py** to validate your implementation.

Step 3: Main Function

In **blackjack.py**, implement a main function that is called when the module is executed as a program (and not imported). The program should take three arguments:

```
usage: blackjack.py <num-simulations> <stand-on-value (1-20)> <'soft'|'hard'>
```

The first argument is an integer specifying the number of simulations to run (should be greater than zero). The second value is an integer between 1 and 20 that indicates the stand-on value. The last argument is either the word "soft" or "hard," indicating which strategy to employ.

At the top of the main function, perform basic validation on the arguments and raise a **ValueError** exception if anything is incorrect. This is also a good place to convert the arguments from **str** to the data type you will use in the rest of the program.

The rest of the main function is a loop for "num-simulations" that initializes a list with two cards (use **get_card** function) and then keeps adding cards to the list according to the return value of the **stand** function implemented above. The output of the program is the percentage of simulations for which the hand was bust. For example:

```
> python3 blackjack.py 10000 12 hard
0.0
```

Upload

Please combine **blackjack.py** and **test_blackjack.py** into a single ZIP file.