Week 4 Assignment: Blackjack Strategy Comparison

Overview

In this assignment, you enhance your code from Week 3's assignment to calculate hand totals for all possible Blackjack strategies of the type presented in that assignment.

Please follow the steps outlined below.

Preparation

Copy **blackjack.py** from Week 3 to a new file called **blackjack2.py**; also copy your test cases from **test_blackjack.py** to **test_blackjack2.py**. Adjust **test_blackjack2.py** so it uses the code defined in **blackjack2.py**.

Step 1: Score namedtuple

In **blackjack2.py**, use **namedtuple** to create a new type called **Score** that has two attributes, **total** and **soft_ace_count**. Adjust the code in your **score** function to return this instead of the generic tuple used in Week 3.

Adjust the test cases in **test_blackjack2.py** as necessary and make sure all existing test cases pass and the main function code works as before.

Step 2: Stand namedtuple

In **blackjack2.py**, use **namedtuple** to create a new type called **Stand** that has two attributes, **stand** and **total**. The "stand" attribute will be used as a Boolean indicating whether to stand or not. The "total" attribute represents the final total score. Adjust the code in your **stand** function to return this instead of the Boolean used in Week 3.

Adjust the test cases in **test_blackjack2.py** as necessary and make sure all existing test cases pass and the main function code works as before.

Step 3: Play Hand Function

In **blackjack2.py**, refactor the game play logic from Week 3 into a function defined as shown below. This function encapsulates the logic of playing a single hand and returns the final score. If the score is 22 or greater (i.e., bust), just return the value 22; otherwise, return the actual final total of the hand. The parameters of the **play_hand** function are the values read from the command line, as in Week 3. After doing this, adjust the main function code to utilize this function and make sure it works as before (i.e., as it was described for Week 3).

```
def play_hand(stand_on_value, stand_on_soft):
      ## TODO: fill this in
      return total
```

Step 4: Refactor Main

In **blackjack2.py**, refactor the functionality so that it takes a single command line argument, which is the number of simulations to run per strategy.

```
> python3 blackjack2.py 100000
```

The updated function should loop across all "stand on values" from 13–20 and both stand on soft and stand on hard, and run the specified number of simulations for each strategy, keeping track of the number of simulations that result in each hand's total value. For example, when the stand-on value is 13, and stand on soft is true, there will be, say, 100K simulations. Use a **defaultdict** to count the number of hands that result in each total score between 13 and 22 using the **play_hand** function defined earlier. Your code should produce output using the **csv.writer** object that corresponds to the table shown below. The rows correspond to the different strategies (stand on hard 13, stand on soft 13, etc.). The columns correspond to the **percentage** of all simulations that resulted in a total of that value. The "Bust" column will have the count for all hands greater than 21. Your code should not have zeros for all the cells.

STRATEGY	13	14	15	16	17	18	19	20	21	BUST
H13	0	0	0	0	0	0	0	0	0	0
S13	0	0	0	0	0	0	0	0	0	0
H14	0	0	0	0	0	0	0	0	0	0
S14	0	0	0	0	0	0	0	0	0	0
H15	0	0	0	0	0	0	0	0	0	0
S15	0	0	0	0	0	0	0	0	0	0
H16	0	0	0	0	0	0	0	0	0	0
S16	0	0	0	0	0	0	0	0	0	0
H17	0	0	0	0	0	0	0	0	0	0
S17	0	0	0	0	0	0	0	0	0	0
H18	0	0	0	0	0	0	0	0	0	0
S18	0	0	0	0	0	0	0	0	0	0
H19	0	0	0	0	0	0	0	0	0	0
S19	0	0	0	0	0	0	0	0	0	0
H20	0	0	0	0	0	0	0	0	0	0
S20	0	0	0	0	0	0	0	0	0	0

Upload

Please combine **blackjack2.py** and **test_blackjack2.py** into a single ZIP file.