Learning how to count cards in BlackJack

Sarah E. Abney, Biosystems Analytics

"Winner, Winner! Chicken dinner!"

Gambling to some is a risk, and to others is a live probability computation. Whether you are the former or the latter.

In this exercise you will learn how to:

- Count BlackJack deals
- Set your deck size
- Have fun!!!

Start off on the road to winning by creating a program called **casinocounter.py**

When you run `-h | --h` on your program you should get the following output:

```
usage: casinocounter.py [-h] [-d int] [-b int] [-r float] str

Counting Cards

positional arguments:
  str                   Three cards that have been dealt for player(P) and
                        Dealer(D): PPD

optional arguments:
  -h, --help            show this help message and exit
  -d int, --decks int   How many decks do you want to start with? (default: 4)
  -b int, --betunit int
                        The amount of currency on the betting chip (default:
                        50)
  -r float, --runningc float
                        Running count (default: 0)
```

Since BlackJack is truly dependent on the card you were dealt. Although you cant count chance, you can count probability of winning! The stakes are low in this program as we are removing the aspect of other players. You and the dealer are playing.

The most popular counting system is the High-Low count, which we use in our code.Meaning that each card will have the following count:

- High = Cards 10, J, K, Q, A = count value of -1 = good for the **player**
- Low = Cards 2, 3, 4, 5, 6 = count value of +1 = good for the **dealer**

- Neutral = Cards 7, 8, 9 = count value of 0 = no good for the **player** OR **dealer**

To give some context, as HIGH cards are played, and therefore depleated from the deck, player advantage DECREASES. In the same respect, as LOW cards are played, player advantage INCREASES - there are less of these that can hurt you in future deals.

Because casinos have caught onto the art of counting cards, more than one deck isused in play. We will need to keep tally of our cards in play, which is called the running count. Based on the number of decks we have in play we can determine the true count which will be important to calculate your chances of winning. The true count is the running count/# of decks remaining(rounded to the nearest half deck). Player advantage INCREASES with HIGHER true count.

For this program we will need:

- An arguement called `-d | --decks`. The default value will be **4** as this is the casino standard (don't ask me how I know this...)
- Our program to accept the cards that were dealt, 2 for the player(**P**) and 1 for the dealer(**D**) in a string like `PPD`

The player should only be able to play card values. The program should present a usage error if the input is not a valid card value:

```
./casinocounter.py DFG
usage: casinocounter.py [-h] [-d int] [-b int] [-r float] str
casinocounter.py: error: Bad input: Stop! Are you even playing cards? "DFG" are not
valid choices. Choose from "2,3,4,5,6,7,8,9,X,J,Q,K,A".
```

Now what do you even do with all these numbers? Well true count can help you determine how much to bet as you play the game. This is determined by using betting units, for simplicity sake this will be an optional arguement `-b | --betunit` with a default of 50 (this can represent any currency). A smart bet is calculated by `true count` minus `1 betting count` for example: You have a true count of 3. 3-1 = 1 betting unit 2 x $50 = $100 should be your bet

We want to win money not lose it! Safe bets are the best bet.

Want to play for multiple rounds? Make sure to use the running count optional flag `-r | --runningc` with your next hand to keep up with your odds (hint: they increase as you play). Make sure that this flag allows for decimals as well!

Try it out and see how your program goes. Maybe try it out in the casino to double down on the code in a real life trial.

When you're ready to gamble run make test. **Real winners** should see:

```
$ make test
pytest -xv test.py
============================== test session starts ===============================
platform linux -- Python 3.6.9, pytest-5.3.5, py-1.8.1, pluggy-0.13.1 --
/usr/bin/python3
cachedir: .pytest_cache
rootdir: /mnt/c/Users/snare/documents/biosystems-analytics-2020/assignments/project
collected 7 items

test.py::test_exists PASSED                                                  [ 14%]
test.py::test_usage PASSED                                                   [ 28%]
test.py::test_bad_string PASSED                                              [ 42%]
test.py::test_input1_238 PASSED                                              [ 57%]
test.py::test_input2_2JK PASSED                                              [ 71%]
test.py::test_input3_runningcflag PASSED                                     [ 85%]
test.py::test_input4_runningcflagD PASSED                                    [100%]

=============================== 7 passed in 1.03s ================================
```