DS 3000 HW 1

Due: Friday Jan 19th @ 11:59 PM EST

Submission Instructions

Submit this `ipynb` file and the a `PDF` file included with the coding results to Gradescope (this can also be done via the assignment on Canvas). To ensure that your submitted files represent your latest code, make sure to give a fresh `Kernel > Restart & Run All` just before uploading the files to gradescope.

Tips for success

- Start early
- Make use of Piazza
- Make use of Office hour
- Remember to use cells and headings to make the notebook easy to read (if a grader cannot find the answer to a problem, you will receive no points for it)
- Under no circumstances may one student view or share their ungraded homework or quiz with another student (see also), though you are welcome to **talk about** (not show each other) the problems.

Part 1 (20 points):

Use the markdown language below to create your own brief wikipedia-esque description of any subject of interest.

Your mini-wiki page must include:

- three headers: a title, subtitle and subsubtitle (the #, ##, ### syntax)
- an embedded image from a web address (use an image hosting site if you'd like to upload your own)
- a table of size at least 3 rows x 3 columns (needn't be correct, but must make sense)
- a list
- a link to another website

Please be brief in your text. Aim for roughly 3 sentences total of text. We won't grade based on content, but keep it appropriate for class. If you make the grader smile, no extra credit will be awarded beyond the satisfaction of having made somebody's day better :)

Nicholas Wong

Computer Science Student @ Northeastern University

Mini-wiki

My name is Nicholas Wong, I am a junior at Northeastern University Studying computer science with a concentration in Artificial Intelligence. I enjoy play sports, watching sports as well as playing board games. I also enjoy coding in my free time, Click here to view my projects on Github.

Times Table

x	1	2	3	4
1	1	2	3	4
2	2	4	6	8
3	3	6	9	12

Embedded Image from the Web

Image

To-Do List

1. Download Python
2. Download Anaconda
3. Open hw1 file into Vscode
4. Complete hw1
5. Save hw1 file as a pdf
6. subit hw1 on gradescope

Part 2.1 (25 points)

In blackjack, players are given two cards from a standard deck of cards and add their values together. Values are assigned to cards as:

- number cards use their own value
- Jack, Queen, King cards all have a value of 10
- aces, **in this problem** will always have a value of 1

Players may then choose to take as many cards as they'd like, one at a time, to increase the sum of their hand's value. The goal is to produce a value as high as possible without exceeding 21. When a player's hand sum increases beyond 21 they are said to 'bust' and they lose the game.

The key to playing well is knowing when to stop taking cards. We'll explore this issue by writing a program to have the computer play many hands of blackjack and count how often they bust.

Examine the function `draw_value()` below which accepts no inputs and returns the value of some card drawn from a standard deck of cards. Pay attention to what `random.choices()`, and its `weights` parameter are doing here, as they **may not be used correctly, and you may have to correct them** .

Assume that you can use this function to simulate playing blackjack. This would mean you draw cards with replacement, so that its possible to draw the same card many times. (In practice, this isn't far from the truth as real casinos often draw from a pile of cards containing multiple, shuffled decks to mitigate the effectiveness of card counting).

In [2]:

```python
# first import the random module to use choices
import random
from collections import defaultdict

# create the function
def draw_value():
    '''draws a single card value

    Args:
        none

    Returns:
        draw (int): choices(values, weights) #the weights are the number of each amount o
f values in a deck
    '''
    values = [1,2,3,4,5,6,7,8,9,10,10,10,10] # include 10 for J, Q, K
    weights = [4,4,4,4,4,4,4,4,4,4,4,4,4]

    draw = random.choices(values, weights=weights)[0]

    return draw

card_count = defaultdict(int)
```

```
for idx in range(52000):
    card = draw_value()
    card_count[card] += 1


print(card_count)

print('The result show that draw_value() works because the results atfer running it 52000 times show that the card is directly porportional to the weight given in the funciton')
```

```
defaultdict(<class 'int'>, {9: 3997, 3: 3964, 6: 4002, 10: 16073, 8: 4050, 2: 3963, 1: 38
66, 4: 4040, 5: 4039, 7: 4006})
The result show that draw_value() works because the results atfer running it 52000 times
show that the card is directly porportional to the weight given in the funciton
```

Its challenging to test a function which returns a random value, by design it gives different values! If we sample many cards from `draw_value()`, we'd expect to see that the total number of cards drawn are distributed, roughly, to our expectation.

- **First, correct the issue(s) with the `draw_value()` function from the first part of this problem (5 points)**
- **Sample 52000 values from `draw_value()` (10 points)**
- **record how many times you observe each card in a `defaultdict` named `card_count` (5 points)**
- **print `card_count` (2 points)**
- **write one sentence which explains why your results validate that `draw_value()` works (3 points)**

Part 2.2 (25 pts)

One of the big questions facing blackjack players is, "Given that my hand already has a value of X, what's the probability that the next card will bust my hand?". For every starting hand, we'll estimate this probability by simulating how often a hand goes bust on the next card among many randomly drawn hands.

Note: Please make sure the docstring and/or comments are given in proper format and variable names are informative and in proper style.

- **initialize `bust_given_start` as a defaultdict**
  - **keys are the starting hands**
  - **values will count how many hands went bust**
  - **e.g. `bust_given_start = {19: 100, 15: 42}` indicates that, among all the simulations:**
    - **100 hands starting at 19 went bust**
    - **42 hands starting at 15 went bust**
- **for every `start_hand` from 2 to 20:**
  - **draw a single card via `draw_value()`**
  - **record the hand as a bust if the `start_hand` + `card_val` exceeds 21**
  - **repeat the above two steps so that you simulate 10,000 total hands**
- **divide all the values in `bust_given_start` by 10,000**
  - **this normalization allows us to interpret the values of the dictionary as probabilities of going bust**
- **print `bust_given_start` so it may be observed**
  - **print in increasing order of `start_hand`**
  - **Notice that even though we tested `start_hand<=11` many times, none of them appear in `bust_given_start` (no need to change this)**
    - **Explain why in one or two complete sentences**

In [5]:

```
import random
from collections import defaultdict

#initialize
bust_given_start = defaultdict(int)

times = 10000
```

```python
for start_hand in range(2, 21):
    for _ in range(times):
        card_val = draw_value()
        if start_hand + card_val > 21:
            bust_given_start[start_hand] += 1

for hand in bust_given_start:
    bust_given_start[hand] /= times

'''
#create start_hand
start_hand = range(2,21)


for player_hand in start_hand:
    times_bust = 0
    for i in range(10000):
        card_val = draw_value()
        if player_hand + card_val >= 21:
            times_bust += 1
    bust_given_start[start_hand] = times_bust
'''

print(bust_given_start)
print('When printing bust_given_start, there is no value less than 12 because it is impos
sible to bust after drawing a card with a starting hand of less than 12. For instance, th
e largest value of a card you can draw is 11 so if you draw a card with the maximum value
you would be at 21 which means you did not bust. ')
```

```
defaultdict(<class 'int'>, {12: 0.2943, 13: 0.3877, 14: 0.4614, 15: 0.535, 16: 0.6135, 17
: 0.7009, 18: 0.7678, 19: 0.8485, 20: 0.9278})
When printing bust_given_start, there is no value less than 12 because it is impossible t
o bust after drawing a card with a starting hand of less than 12. For instance, the large
st value of a card you can draw is 11 so if you draw a card with the maximum value you wo
uld be at 21 which means you did not bust.
```

Part 3 (30 pts)

You are tasked with creating a program that calculates the final grade of a student based on their scores in different subjects. The program should follow these rules:

- **Ask the user for the student's name.(2 points)**
- **Ask the user for the number of subjects the student has taken. (2 points)**
- **For each subject, ask the user to input the subject name and the student's score (out of 100). (10 points)**
- **Calculate the average score for all subjects. (3 points)**
- **Based on the average score, assign a letter grade to the student using the following grading scale: (10 points)**
  - **A: 90-100**
  - **B: 80-89**
  - **C: 70-79**
  - **D: 60-69**
  - **F: 0-59**
- **Display the student's name, average score, and letter grade. (3 points)**

In [6]:

```python
student_name = input('What is your name')
number_subjects = int(input('How many subjects have you taken?'))

subjects = {}

for i in range(number_subjects):
    subject_name = input("What is the name of the subject: ")
    subject_score = float(input("What is your score in {subject_name} (out of 100): "))
    subjects[subject_name] = subject_score
```

```python
total_score = sum(subjects.values())
average_score = total_score / len(subjects)

if 90 <= average_score <= 100:
    letter_grade = 'A'
elif 80 <= average_score < 90:
    letter_grade = 'B'
elif 70 <= average_score < 80:
    letter_grade = 'C'
elif 60 <= average_score < 70:
    letter_grade = 'D'
else:
    letter_grade = 'F'


print("Student Name:", student_name)
print("Average Score:", average_score)
print("Letter Grade:", letter_grade)
```

```
Student Name: nick
Average Score: 92.5
Letter Grade: A
```