Homework 5: Applying Functions and Iteration

```
# Don't change this cell; just run it.

import numpy as np
from datascience import *

# These lines do some fancy plotting magic.
import matplotlib
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
import warnings
warnings.simplefilter('ignore', FutureWarning)
```

1. 2019 Cal Football Season

Shoumik is trying to analyze how well the UC Berkeley ("Cal") football team performed in the 2019 season. A football game is divided into four periods, called quarters. The number of points Cal scored in each quarter, and the number of points their opponent scored in each quarter are stored in a table called `cal_fb.csv`.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Just run this cell
# Read in the cal_fb csv file
games = Table().read_table("/content/drive/MyDrive/cal_fb.csv")
games.show()
```

Opponent	Cal 1Q	Cal 2Q	Cal 3Q	Cal 4Q	Opp 1Q	Opp 2Q	Opp 3Q	Opp 4Q
UC Davis	0	13	7	7	10	0	3	0
Washington	0	3	14	3	0	10	3	6
North Texas	20	0	3	0	0	3	7	7
Ole Miss	7	7	14	0	7	6	0	7
Arizona State	7	0	7	3	7	0	7	10
Oregon	7	0	0	0	0	0	10	7
Oregon State	0	3	14	0	7	7	0	7
Utah	0	0	0	0	7	21	7	0
Washington State	6	7	7	13	5	6	3	6
USC	7	3	0	7	10	7	17	7
Stanford	7	3	0	14	7	3	7	3
UCLA	7	7	7	7	7	3	9	0
Illinois	7	14	7	7	10	3	0	7

Let's start by finding the total points each team scored in a game.

Question 1. Write a function called `sum_scores`. It should take four arguments, where each argument is the team's score for that quarter. It should return the team's total score for that game.

```
def sum_scores(q1, q2, q3, q4):
    '''Returns the total score calculated by adding up the score of each quarter'''
    return q1 + q2 + q3 + q4

sum_scores(14, 7, 3, 0) #DO NOT CHANGE THIS LINE
```

```
24
```

Question 2. Create a new table `final_scores` with three columns in this *specific* order: `Opponent`, `Cal Score`, `Opponent Score`. You will have to create the `Cal Score` and `Opponent Score` columns. Use the function `sum_scores` you just defined in the previous question for this problem.

Hint: If you want to apply a function that takes in multiple arguments, you can pass multiple column names as arguments in `tbl.apply()`. The column values will be passed into the corresponding arguments of the function. Take a look at the python reference for syntax.

Tip: If you're running into issues creating final_scores, check that `cal_scores` and `opponent_scores` output what you want.

```
cal_scores = games.apply(sum_scores, "Cal 1Q","Cal 2Q","Cal 3Q","Cal 4Q")
opponent_scores = games.apply(sum_scores, "Opp 1Q","Opp 2Q","Opp 3Q","Opp 4Q")
Opponent = games.column(0)
final_scores = Table().with_columns(
    'Opponent', Opponent,
    'Cal Score', cal_scores,
    'Opponent Score', opponent_scores
)

final_scores
```

Opponent	Cal Score	Opponent Score
UC Davis	27	13
Washington	20	19
North Texas	23	17
Ole Miss	28	20
Arizona State	17	24
Oregon	7	17
Oregon State	17	21
Utah	0	35
Washington State	33	20
USC	17	41

... (3 rows omitted)

We can get specific row objects from a table. You can use `tbl.row(n)` to get the `n` th row of a table. `row.item("column_name")` will allow you to select the element that corresponds to `column_name` in a particular row. Here's an example:

```
# Just run this cell
# We got the Axe!
games.row(10)
```

```
Row(Opponent='Stanford', Cal 1Q=7, Cal 2Q=3, Cal 3Q=0, Cal 4Q=14, Opp 1Q=7, Opp 2Q=3, Opp 3Q=7, Opp 4Q=3)
```

```
# Just run this cell
games.row(10).item("Cal 4Q")
```

```
14
```

Question 3. We want to see for a particular game whether or not Cal won. Write a function called `did_cal_win`. It should take one argument: a row object from the `final_scores` table. It should return either `True` if Cal's score was greater than the Opponent's score, and `False` otherwise.

```
def did_cal_win(row):
    if(final_scores.column(1)[row-1]>final_scores.column(2)[row-1]):
      return True
    return False

print(did_cal_win(5))
```

```
False
```

Question 4. Shoumik wants to see how Cal did against every opponent during the 2019 season. Using the `final_scores` table, assign `results` to an array of `True` and `False` values that correspond to whether or not Cal won. Add the `results` array to the `final_scores` table, and assign this to `final_scores_with_results`. Then, respectively assign the number of wins and losses Cal had to `cal_wins` and `cal_losses`.

Hint: When you only pass a function name and no column labels through `tbl.apply()`, the function gets applied to every row in `tbl`

```
results = []
for x in range(len(final_scores.column(0))):
  results.append(did_cal_win(x+1))

final_scores_with_results = final_scores.with_column("Results", results)
cal_wins = np.count_nonzero(final_scores_with_results.column(3))
cal_losses = len(final_scores_with_results.column(3)) - np.count_nonzero(final_scores_with_results.column(3))


# Don't delete or edit the following line:
print(f"In the 2019 Season, Cal Football won {cal_wins} games and lost {cal_losses} games. Go Bears!")
```

```
In the 2019 Season, Cal Football won 8 games and lost 5 games. Go Bears!
```

Unrolling Loops

"Unrolling" a `for` loop means to manually write out all the code that it executes. The result is code that does the same thing as the loop, but without the structure of the loop. For example, for the following loop:

```
for num in np.arange(3):
    print("The number is", num)
```

The unrolled version would look like this:

```
print("The number is", 0)
print("The number is", 1)
print("The number is", 2)
```

Unrolling a `for` loop is a great way to understand what the loop is doing during each step. In this exercise, you'll practice unrolling `for` loops.

In each question below, write code that does the same thing as the given code, but with any `for` loops unrolled. It's a good idea to run both your answer and the original code to verify that they do the same thing. (Of course, if the code does something random, you'll get a different random outcome than the original code!)

First, run the cell below to load data that will be used in a few questions. It's a table with 52 rows, one for each type of card in a deck of playing cards. A playing card has a "suit" ("♠", "♣", "♥", or "♦") and a "rank" (2 through 10, J, Q, K, or A). There are 4 suits and 13 ranks, so there are $4 \times 13 = 52$ different cards.

```
deck = Table().read_table("/content/drive/MyDrive/deck1.csv")
deck
```

Rank	Suit
2	♠
2	♣
2	♥
2	♦
3	♠
3	♣
3	♥
3	♦
4	♠
4	♣

... (42 rows omitted)

Question 1. Unroll the code below.

Hint: `np.random.randint` returns a random integer between 0 (inclusive) and the value that's passed in (exclusive).

```
# This table will hold the cards in a randomly-drawn hand of
# 5 cards.  We simulate cards being drawn as follows: We draw
# a card at random from the deck, make a copy of it, put the
# copy in our hand, and put the card back in the deck.  That
# means we might draw the same card multiple times, which is
# different from a normal draw in most card games.
hand = Table().with_columns("Rank", make_array(), "Suit", make_array())
for suit in np.arange(5):
    card = deck.row(np.random.randint(deck.num_rows))
    hand = hand.with_row(card)
hand
```

Rank	Suit
5	♥
Q	♣
7	♠
5	♦
8	♣

```
#Unroll the code here. Creating just the first two rows of the table is fine.
hand = Table().with_columns("Rank", make_array(), "Suit", make_array())
card = deck.row(np.random.randint(deck.num_rows))
hand = hand.with_row(card)
card = deck.row(np.random.randint(deck.num_rows))
hand = hand.with_row(card)
card = deck.row(np.random.randint(deck.num_rows))
hand = hand.with_row(card)
card = deck.row(np.random.randint(deck.num_rows))
hand = hand.with_row(card)
card = deck.row(np.random.randint(deck.num_rows))
hand = hand.with_row(card)
hand
```

Rank	Suit
8	♣
10	♣
2	♣
4	♦
A	♦

Question 2. Unroll the code below.

```
# This table will hold the cards in a randomly-drawn hand of
# 4 cards.  The cards are drawn as follows: For each of the
# 4 suits, we draw a random card of that suit and put it into
# our hand.  The cards within a suit are drawn uniformly at
# random, meaning each card of the suit has an equal chance of
# being drawn.
hand_of_4 = Table().with_columns("Rank", make_array(), "Suit", make_array())
for suit in make_array("♠", "♣", "♥", "♦"):
    cards_of_suit = deck.where("Suit", are.equal_to(suit))
    card = cards_of_suit.row(np.random.randint(cards_of_suit.num_rows))
    hand_of_4 = hand_of_4.with_row(card)
hand_of_4
```

Rank	Suit
8	♠
4	♣
2	♥
8	♦

```
#Unroll the code here. Creating just the first two rows of the table is fine.
hand_of_4 = Table().with_columns("Rank", make_array(), "Suit", make_array())
suit_array = make_array("♠", "♣", "♥", "♦");
cards_of_suit = deck.where("Suit", are.equal_to(suit_array[0]))
card = cards_of_suit.row(np.random.randint(cards_of_suit.num_rows))
hand_of_4 = hand_of_4.with_row(card)
cards_of_suit = deck.where("Suit", are.equal_to(suit_array[1]))
card = cards_of_suit.row(np.random.randint(cards_of_suit.num_rows))
hand_of_4 = hand_of_4.with_row(card)
cards_of_suit = deck.where("Suit", are.equal_to(suit_array[2]))
card = cards_of_suit.row(np.random.randint(cards_of_suit.num_rows))
hand_of_4 = hand_of_4.with_row(card)
cards_of_suit = deck.where("Suit", are.equal_to(suit_array[3]))
card = cards_of_suit.row(np.random.randint(cards_of_suit.num_rows))
hand_of_4 = hand_of_4.with_row(card)

hand_of_4
```

Rank	Suit
10	♠
7	♣
7	♥
6	♦

3. Submission

+ Code + Text

This lab is due by **Monday, March 4th at 11:59pm**.

Once your assignment is complete, do two things:

1. Change the file name by by adding “Submitted” at the end. For example, if I were a student submitting lab01, the file would initially be called “lab01Liz.” After completing it, I would change the name to “lab01LizSubmitted.”
2. Print your notebook as a pdf. Go to File -> Print -> save as pdf. Then upload the pdf to the corresponding assignment (lab/hw) on Canvas.