Homework 5: Applying Functions and Iteration

Please complete this notebook by filling in the cells provided. Before you begin, execute the following cell to load the provided tests. Each time you start your server, you will need to execute this cell again to load the tests.

Homework 5 is due Thursday, 2/27 at 11:59pm. You will receive an early submission bonus point if you turn in your final submission by Wednesday, 2/26 at 11:59pm. Late work will not be accepted as per the policies (http://data8.org/sp20/policies.html) of this course.

Throughout this homework and all future ones, please be sure to not re-assign variables throughout the notebook! For example, if you use `max_temperature` in your answer to one question, do not reassign it later on. Moreover, please be sure to only put your written answers in the provided cells.

In [1]:
```python
# 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)

from client.api.notebook import Notebook
ok = Notebook('hw05.ok')
_ = ok.auth(inline=True)
```

```
=====================================================================
Assignment: Homework 5: Applying Functions and Iteration
OK, version v1.14.19
=====================================================================

Successfully logged in as tvilayth@berkeley.edu
```

1. 2019 Cal Football Season

Shoumik is trying to analyze how well the 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`.

In [2]:
```
# Just run this cell
# Read in the cal_fb csv file
games = Table().read_table("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.

```
BEGIN QUESTION
name: q1_1
manual: false
```

In [3]:
```
""" # BEGIN PROMPT
def sum_scores(...):
    '''Returns the total score calculated by adding up the score of each quarter'''
    ...
"""; # END PROMPT
# BEGIN SOLUTION NO PROMPT
def sum_scores(q1, q2, q3, q4):
    return q1 + q2 + q3 + q4
# END SOLUTION

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

Out[3]:
```
24
```

```
In [4]: # TEST
        sum_scores(2, 3, 6, 1)

Out[4]: 12
```

```
In [5]: # TEST
        sum_scores(-2,3,5,-10)

Out[5]: -4
```

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.

```
BEGIN QUESTION
name: q1_2
manual: false
```

In [6]:
```
cal_scores = games.apply(sum_scores, "Cal 1Q","Cal 2Q", "Cal 3Q", "Cal 4
Q") # SOLUTION
opponent_scores = games.apply(sum_scores, "Opp 1Q", "Opp 2Q", "Opp 3Q",
"Opp 4Q") # SOLUTION
final_scores = games.with_columns("Cal Score", cal_scores, "Opponent Sco
re", opponent_scores).select("Opponent", "Cal Score", "Opponent Score")
# SOLUTION
final_scores
```

Out[6]:

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)

In [7]:
```
# TEST
final_scores.num_columns
```

Out[7]:
```
3
```

In [8]:
```
# TEST
set(['Opponent', 'Cal Score', 'Opponent Score']) == set(final_scores.lab
els)
```

Out[8]:
```
True
```

In [9]:
```
# HIDDEN TEST
final_scores.take(range(5, 11))
```

Out[9]:

Opponent	Cal Score	Opponent Score
Oregon	7	17
Oregon State	17	21
Utah	0	35
Washington State	33	20
USC	17	41
Stanford	24	20

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:

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

Out[10]: Row(Opponent='Stanford', Cal 1Q=7, Cal 2Q=3, Cal 3Q=0, Cal 4Q=14, Opp 1
         Q=7, Opp 2Q=3, Opp 3Q=7, Opp 4Q=3)
```

```
In [11]: # Just run this cell
         games.row(10).item("Cal 4Q")

Out[11]: 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.

```
BEGIN QUESTION
name: q1_3
manual: false
```

```
In [12]: """ # BEGIN PROMPT
         def did_cal_win(row):
             ...
         """; # END PROMPT
         # BEGIN SOLUTION NO PROMPT
         def did_cal_win(row):
             return row.item("Cal Score") > row.item("Opponent Score")
         # END SOLUTION
```

```
In [13]: # TEST
         'did_cal_win' in globals()

Out[13]: True
```

```
In [14]: # TEST
         did_cal_win(final_scores.row(1))

Out[14]: True
```

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`

```
BEGIN QUESTION
name: q1_4
manual: false
```

In [15]:
```
results = final_scores.apply(did_cal_win) # SOLUTION
final_scores_with_results = final_scores.with_columns("Results", results
) # SOLUTION
cal_wins = final_scores_with_results.where("Results", True).num_rows # S
OLUTION
cal_losses = final_scores_with_results.num_rows - cal_wins # SOLUTION

# 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 Bear
s!
```

In [16]:
```
# TEST
0 <= cal_wins <= 10
```

Out[16]:
```
True
```

In [17]:
```
# TEST
0 <= cal_losses <= 10
```

Out[17]:
```
True
```

In [18]:
```
# HIDDEN TEST
cal_wins - cal_losses
```

Out[18]:
```
3
```

In [19]:
```
# HIDDEN TEST
[results.item(6), results.item(1), results.item(5)] == [False, True, Fal
se]
```

Out[19]:
```
True
```

You've reached the end of the required questions for this assignment! Please submit your work by saving your notebook and running the submit cell below.

Continue on to the optional section for some practice with iterations and for loops!

```
In [20]: _ = ok.submit()
```

```
Saving notebook...

---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-20-cc46ca874451> in <module>
----> 1 _ = ok.submit()

/opt/anaconda3/lib/python3.7/site-packages/client/api/notebook.py in submit(self)
     69         messages = {}
     70         self.assignment.set_args(submit=True)
---> 71         if self.save(messages):
     72             return self.run('backup', messages)
     73         else:

/opt/anaconda3/lib/python3.7/site-packages/client/api/notebook.py in save(self, messages, delay, attempts)
     78 
     79     def save(self, messages, delay=0.5, attempts=3):
---> 80         saved = self.save_notebook()
     81         if not saved:
     82             return None

/opt/anaconda3/lib/python3.7/site-packages/client/api/notebook.py in save_notebook(self)
    115         # Wait for first .ipynb to save
    116         if ipynbs:
--> 117             if wait_for_save(ipynbs[0]):
    118                 print("Saved '{}'.".format(ipynbs[0]))
    119             else:

/opt/anaconda3/lib/python3.7/site-packages/client/api/notebook.py in wait_for_save(filename, timeout)
    160     Returns True if a save was detected, and False otherwise.
    161     """
--> 162     modification_time = os.path.getmtime(filename)
    163     start_time = time.time()
    164     while time.time() < start_time + timeout:

/opt/anaconda3/lib/python3.7/genericpath.py in getmtime(filename)
     53 def getmtime(filename):
     54     """Return the last modification time of a file, reported by os.stat()."""
---> 55     return os.stat(filename).st_mtime
     56 
     57 

FileNotFoundError: [Errno 2] No such file or directory: 'hw05.ipynb'
```

(Optional) Unrolling Loops

The rest of this homework is optional. Do it for your own practice, but it will not be incorporated into the final grading!

"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.

In [21]:
```
deck = Table.read_table("deck.csv")
deck
```

Out[21]:

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

... (42 rows omitted)

Optional 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).

```
BEGIN QUESTION
name: q2_1
manual: false
```

In [22]:
```
# 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
```

Out[22]:

Rank	Suit
8	♣
2	♦
9	♦
J	♣
8	♦

In [23]:
```
hand = Table().with_columns("Rank", make_array(), "Suit", make_array())
""" # BEGIN PROMPT
...
"""; # END PROMPT
# BEGIN SOLUTION NO PROMPT
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
# END SOLUTION
```

Out[23]:

Rank	Suit
5	♠
5	♣
2	♣
4	♣
5	♥

Optional Question 2. Unroll the code below.

```
BEGIN QUESTION
name: q2_2
manual: false
```

In [24]:
```
for joke_iteration in np.arange(4):
    print("Knock, knock.")
    print("Who's there?")
    print("Banana.")
    print("Banana who?")
print("Knock, knock.")
print("Who's there?")
print("Orange.")
print("Orange who?")
print("Orange you glad I didn't say banana?")
```

```
Knock, knock.
Who's there?
Banana.
Banana who?
Knock, knock.
Who's there?
Banana.
Banana who?
Knock, knock.
Who's there?
Banana.
Banana who?
Knock, knock.
Who's there?
Banana.
Banana who?
Knock, knock.
Who's there?
Orange.
Orange who?
Orange you glad I didn't say banana?
```

In [25]:

```
""" # BEGIN PROMPT
...
"""; # END PROMPT
# BEGIN SOLUTION NO PROMPT
print("Knock, knock.")
print("Who's there?")
print("Banana.")
print("Banana who?")
print("Knock, knock.")
print("Who's there?")
print("Banana.")
print("Banana who?")
print("Knock, knock.")
print("Who's there?")
print("Banana.")
print("Banana who?")
print("Knock, knock.")
print("Who's there?")
print("Banana.")
print("Banana who?")
print("Knock, knock.")
print("Who's there?")
print("Orange.")
print("Orange who?")
print("Orange you glad I didn't say banana?")
# END SOLUTION
```

```
Knock, knock.
Who's there?
Banana.
Banana who?
Knock, knock.
Who's there?
Banana.
Banana who?
Knock, knock.
Who's there?
Banana.
Banana who?
Knock, knock.
Who's there?
Banana.
Banana who?
Knock, knock.
Who's there?
Orange.
Orange who?
Orange you glad I didn't say banana?
```

Optional Question 3. Unroll the code below.

```
BEGIN QUESTION
name: q2_3
manual: false
```

In [26]:
```
# 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_arra
y())
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
```

Out[26]:

Rank	Suit
5	♠
3	♣
9	♥
J	♦

In [27]:
```
hand_of_4 = Table().with_columns("Rank", make_array(), "Suit", make_arra
y())
""" # BEGIN PROMPT
...
"""; # END PROMPT
# BEGIN SOLUTION NO PROMPT
cards_of_suit = deck.where("Suit", are.equal_to("♠"))
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("♣"))
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("♥"))
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("♦"))
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
# END SOLUTION
```

Out[27]:

Rank	Suit
2	♠
K	♣
9	♥
7	♦

3. Submission

Once you're finished, select "Save and Checkpoint" in the File menu and then execute the `submit` cell below. The result will contain a link that you can use to check that your assignment has been submitted successfully. If you submit more than once before the deadline, we will only grade your final submission. If you mistakenly submit the wrong one, you can head to okpy.org (https://okpy.org/) and flag the correct version. To do so, go to the website, click on this assignment, and find the version you would like to have graded. There should be an option to flag that submission for grading!

```
In [28]: _ = ok.submit()
```

```
Saving notebook...

---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-28-cc46ca874451> in <module>
----> 1 _ = ok.submit()

/opt/anaconda3/lib/python3.7/site-packages/client/api/notebook.py in submit(self)
     69         messages = {}
     70         self.assignment.set_args(submit=True)
---> 71         if self.save(messages):
     72             return self.run('backup', messages)
     73         else:

/opt/anaconda3/lib/python3.7/site-packages/client/api/notebook.py in save(self, messages, delay, attempts)
     78 
     79     def save(self, messages, delay=0.5, attempts=3):
---> 80         saved = self.save_notebook()
     81         if not saved:
     82             return None

/opt/anaconda3/lib/python3.7/site-packages/client/api/notebook.py in save_notebook(self)
    115         # Wait for first .ipynb to save
    116         if ipynbs:
--> 117             if wait_for_save(ipynbs[0]):
    118                 print("Saved '{}'.".format(ipynbs[0]))
    119             else:

/opt/anaconda3/lib/python3.7/site-packages/client/api/notebook.py in wait_for_save(filename, timeout)
    160     Returns True if a save was detected, and False otherwise.
    161     """
--> 162     modification_time = os.path.getmtime(filename)
    163     start_time = time.time()
    164     while time.time() < start_time + timeout:

/opt/anaconda3/lib/python3.7/genericpath.py in getmtime(filename)
     53 def getmtime(filename):
     54     """Return the last modification time of a file, reported by os.stat()."""
---> 55     return os.stat(filename).st_mtime
     56 
     57 

FileNotFoundError: [Errno 2] No such file or directory: 'hw05.ipynb'
```

In [29]:
```
# For your convenience, you can run this cell to run all the tests at on
ce!
import os
print("Running all tests...")
_ = [ok.grade(q[:-3]) for q in os.listdir("tests") if q.startswith('q')
and len(q) <= 10]
print("Finished running all tests.")
```

```
Running all tests...
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed

Finished running all tests.
```