hw05_master

October 11, 2019

1 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, 10/3 at 11:59pm. You will receive an early submission bonus point if you turn in your final submission by Wednesday, 10/2 at 11:59pm. Late work will not be accepted as per the policies 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.

[ ]:
```
# 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)
```

1.1 1. Counting Calories Burned from Exercise

Suppose you'd like to count how many calories you've burned from exercise. You do 4 kinds of exercise: yoga, walking, sprinting, and volleyball. Every day in January, you record how many minutes of each kind of exercise you did that day. Those data are in a table called `exercise.csv`.

[2]:
```
exercise = Table.read_table('exercise.csv')
exercise
```

[2]:
```
Day (in January) | yoga | walking | sprinting | volleyball
1                | 50   | 0       | 0         | 0
2                | 0    | 65      | 0         | 65
3                | 50   | 70      | 0         | 0
4                | 0    | 0       | 0         | 75
5                | 0    | 0       | 0         | 0
6                | 0    | 0       | 0         | 0
7                | 0    | 0       | 0         | 0
8                | 0    | 55      | 0         | 0
9                | 0    | 0       | 0         | 0
10               | 0    | 30      | 0         | 0
… (21 rows omitted)
```

Different forms of exercise burn calories at different rates; for example, sprinting is more vigorous than walking. The table `calories_per_minute` contains estimates of the calories per minute burned by each activity.

[3]:
```
calories_per_minute = Table.read_table('calories_per_minute.csv')
calories_per_minute
```

[3]:
```
Exercise   | Calories per minute
yoga       | 6
walking    | 5
sprinting  | 20
volleyball | 4
```

Let's start by finding the total number of minutes you spent exercising each day.

Question 1. Write a function called `compute_exercise_time`. It should take one argument, a row from the `exercise` table that contains the day and amounts of time (in minutes) spent on yoga, walking, sprinting, and volleyball. It should return the total time spent exercising.

Hint You can `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.

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

[4]:
```
""" # BEGIN PROMPT
def compute_exercise_time(exercise_row):
    ...
"""; # END PROMPT
# BEGIN SOLUTION NO PROMPT
def compute_exercise_time(exercise_row):
    yoga_time = exercise_row.item("yoga")
```

```
    walking_time = exercise_row.item("walking")
    sprinting_time = exercise_row.item("sprinting")
    volleyball_time = exercise_row.item("volleyball")
    return yoga_time + walking_time + sprinting_time + volleyball_time
# END SOLUTION

compute_exercise_time(exercise.row(0))
```

[4]: 50.0

Question 2. Create a new table `exercise_time` that is a copy of the `exercise` table, with a new column called `Total Exercise Time` that describes the total time (in minutes) spent exercising on each day.

Hint: When you only pass a function name through `tbl.apply()`, the function gets applied to every row in `tbl`

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

[7]:
```
exercise_time = exercise.with_columns("Total Exercise Time", exercise.
 ↪apply(compute_exercise_time)) # SOLUTION
exercise_time
```

[7]:

Day (in January)	yoga	walking	sprinting	volleyball	Total Exercise Time
1	50	0	0	0	50
2	0	65	0	65	130
3	50	70	0	0	120
4	0	0	0	75	75
5	0	0	0	0	0
6	0	0	0	0	0
7	0	0	0	0	0
8	0	55	0	0	55
9	0	0	0	0	0
10	0	30	0	0	30

… (21 rows omitted)

To compute the calories you've burned on a particular day, you multiply the time spent on each kind of exercise by the calories burned per minute by that exercise, then add up those 4 numbers.

Question 3. Write a function called `compute_calories`. It should take 4 arguments, the amounts of time (in minutes) spent on yoga, walking, sprinting, and volleyball, respectively. It should return the total number of calories burned.

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

[18]:
```
""" # BEGIN PROMPT
def compute_calories(..., ..., ..., ...)
    ...
"""; # END PROMPT
# BEGIN SOLUTION NO PROMPT
def compute_calories(yoga_time, walking_time, sprinting_time, volleyball_time):
    return sum(make_array(yoga_time, walking_time, sprinting_time,␣
 ↪volleyball_time) * calories_per_minute.column("Calories per minute"))
# END SOLUTION
```

Question 4. Make a table called `exercise_with_totals` that's a copy of `exercise`, but with an additional column called "Total calories burned exercising". That column should contain the total number of calories burned from exercise on each day. Compute that column using `apply` and your `compute_calories` function.

Hint: If you want to apply a function that takes multiple arguments, you can pass through multiple column names as arguments in `tbl.apply()` to call your function on all corresponding columns.

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

[21]:
```
exercise_with_totals = exercise.with_column("Total calories burned exercising",␣
 ↪exercise.apply(compute_calories, "yoga", "walking", "sprinting",␣
 ↪"volleyball")) # SOLUTION
exercise_with_totals
```

[21]:

Day (in January)	yoga	walking	sprinting	volleyball	Total calories burned exercising
1	50	0	0	0	300
2	0	65	0	65	585
3	50	70	0	0	650
4	0	0	0	75	300
5	0	0	0	0	0
6	0	0	0	0	0
7	0	0	0	0	0
8	0	55	0	0	275
9	0	0	0	0	0
10	0	30	0	0	150

… (21 rows omitted)

Question 5. How many calories were burned from exercise in the 2nd week of January (that is, on days 8 through 14, inclusive)? Call that number `calories_burned`. Compute the answer with code, not by looking at the data.

```
BEGIN QUESTION
name: q1_5
manual: false
```

[25]:
```
calories_burned = sum(exercise_with_totals.where("Day (in January)", are.
↪between_or_equal_to(8, 14)).column("Total calories burned exercising"))␣
↪#SOLUTION
calories_burned
```

[25]:
```
2715.0
```

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!

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

1.2 (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.

[3]:
```
deck = Table.read_table("deck.csv")
deck
```

[3]:
```
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
```

[4]:
```
# 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
```

[4]:
```
Rank | Suit
A    |
6    |
A    |
5    |
4    |
```

[5]:
```
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
```

[5]:
```
Rank | Suit
10   |
9    |
6    |
9    |
8    |
```

Optional Question 2. Unroll the code below.

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

[8]:
```
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?
```

[9]:
```
""" # 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
```

[10]:
```
# 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
```

[10]:
```
Rank | Suit
5    |
8    |
7    |
9    |
```

[11]:
```
hand_of_4 = Table().with_columns("Rank", make_array(), "Suit", make_array())
""" # 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
```

[11]:
```
Rank | Suit
9    |
4    |
8    |
A    |
```

1.3 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 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!

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

[ ]:
```
# For your convenience, you can run this cell to run all the tests at once!
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.")
```