hw05

September 25, 2024

```
[66]: # Initialize Otter
      import otter
      grader = otter.Notebook("hw05.ipynb")
```

1 Homework 5: Applying Functions and Iteration

Please complete this notebook by filling in the cells provided. Before you begin, execute the previous cell to load the provided tests.

Helpful Resource: - Python Reference: Cheat sheet of helpful array & table methods used in Data 8!

Recommended Readings:

- Tabular Thinking Guide
- Applying Functions
- Conditionals
- Iteration

Please complete this notebook by filling in the cells provided. Before you begin, execute the following cell to setup the notebook by importing some helpful libraries. Each time you start your server, you will need to execute this cell again.

For all problems that you must write explanations and sentences for, you **must** provide your answer in the designated space. **Moreover, 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. Otherwise, you will fail tests that you thought you were passing previously!

Note: This homework has hidden tests on it. That means even though the tests may say 100% passed, it doesn't mean your final grade will be 100%. We will be running more tests for correctness once everyone turns in the homework.

Directly sharing answers is not okay, but discussing problems with the course staff or with other students is encouraged.

You should start early so that you have time to get help if you're stuck.

1.1 1. 2021 Cal Football Season

[67]:
```
# Run this cell to set up the notebook, but please don't change it.

# These lines import the numpy and datascience modules.
import numpy as np
from datascience import *
import warnings
warnings.simplefilter('ignore', FutureWarning)
```

James is trying to analyze how well the Cal football team performed in the 2021 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`.

[68]:
```
# Just run this cell
# Read in the cal_fb csv file
games = Table().read_table("cal_fb.csv")
games.show()
```

```
<IPython.core.display.HTML object>
```

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 represents integers corresponding to the team's score for each quarter. It should return the team's total score for that game. **(20 Points)**

Note: Don't overthink this question!

[69]:
```
def sum_scores(*args):
    '''Returns the total score calculated by adding up the score of each␣
 ↪quarter'''
    return sum(args)

# # or
# def sum_scores(q1, q2, q3, q4):
#     '''Returns the total score calculated by adding up the score of each␣
 ↪quarter'''
#     return sum([q1, q2, q3, q4])

# or
# 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
```

```
[69]: 24
```

```
[70]: grader.check("q1_1")
```

```
[70]: q1_1 results: All test cases passed!
```

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. **(20 Points)**

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 Sheet and Lecture 13's demo for syntax.

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

```
[92]: cal_scores = games.select([1, 2, 3, 4])
      # cal_scores

      opponent_scores = games.select([5, 6, 7 ,8])
      # opponent_scores

      # 'select' creates a new table, 'with_columns' adds new columns
      # 'apply' applies the funct sum_scores to each row of the data cal_scores
      final_scores = games.select('Opponent').with_columns\
          (
          'Cal Score', games.apply(sum_scores, cal_scores),\
          'Opponent Score', games.apply(sum_scores, opponent_scores)
          )

      final_scores
```

```
[92]: Opponent          | Cal Score | Opponent Score
      Nevada            | 17        | 22
      TCU               | 32        | 34
      Sacramento State  | 42        | 30
      Washington        | 24        | 31
      Washington State  | 6         | 21
      Oregon            | 17        | 24
      Colorado          | 26        | 3
      Oregon State      | 39        | 25
      Arizona           | 3         | 10
      Stanford          | 41        | 11
      … (2 rows omitted)
```

[93]:
```
grader.check("q1_2")
```

[93]:
```
q1_2 results: All test cases passed!
```

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:

[94]:
```
# Just run this cell
# We got the Axe!
games.row(9)
```

[94]:
```
Row(Opponent='Stanford', Cal 1Q=0, Cal 2Q=14, Cal 3Q=13, Cal 4Q=14, Opp 1Q=0,
Opp 2Q=3, Opp 3Q=0, Opp 4Q=8)
```

[95]:
```
# Just run this cell
games.row(9).item("Cal 4Q")
```

[95]:
```
14
```

Question 3. We want to see for a particular game whether or not Cal lost. Write a function called `did_cal_lose`. It should take one argument: a **row object** from the `final_scores` table. It should return either `True` if Cal's score was less than the Opponent's score, and `False` otherwise. **(20 Points)**

Note 1: "Row object" means a row from the table that contains all the data for that specific row. It is **not** the index of a row. Do not try and call `final_scores.row(row)` inside of the function.

Note 2: If you're still confused by row objects, try printing out `final_scores.row(1)` in a new cell to visually see what it looks like! This piece of code is pulling out the row object located at index 1 of the `final_scores` table and returning it. When you display it in a cell, you'll see that it is not located within a table, but is instead a standalone row object!

[96]:
```
final_scores.row(1)
```

[96]:
```
Row(Opponent='TCU', Cal Score=32, Opponent Score=34)
```

[97]:
```
final_scores.row(1).item(1)
```

[97]:
```
32
```

[98]:
```
bool(final_scores.row(1).item(1) < final_scores.row(1).item(2))
```

[98]:
```
True
```

[99]:
```
bool(games.apply(sum_scores, cal_scores).item(1) < games.apply(sum_scores,␣
↪opponent_scores).item(1))
```

[99]:
```
True
```

```
[100]: def did_cal_lose(row):
           cal_score_int = row.item(1)
           opp_score_int = row.item(2)
           return bool( cal_score_int < opp_score_int)

       did_cal_lose(final_scores.row(1)) #DO NOT CHANGE THIS LINE
```

```
[100]: True
```

```
[101]: grader.check("q1_3")
```

```
[101]: q1_3 results: All test cases passed!
```

Question 4. James wants to see how Cal did against every opponent during the 2021 season. Using the `final_scores` table:

1. Assign `results` to an array of `True` and `False` values that correspond to whether or not Cal lost.
2. Add the `results` array to the `final_scores` table in a column named `Results`, and assign this to `final_scores_with_results`.
3. Then, respectively assign the number of wins and losses Cal had to `cal_wins` and `cal_losses`.

(20 Points)

Hint 1: `True` and `False` are **not** strings. What data type are they?

Hint 2: `tbl.num_rows` might be helpful too.

Hint 3: When you only pass a function name and no column labels through `tbl.apply()`, the function gets applied to every row in `tbl`.

```
[127]: results = [did_cal_lose(final_scores.row(i)) for i in range(final_scores.
         ↪num_rows)]
       print(results)

       final_scores_with_results = final_scores.with_column('Results', results)
       print(final_scores_with_results)

       cal_losses = sum(results)
       print(cal_losses)
       cal_wins = final_scores.num_rows - cal_losses
       print(cal_wins)

       # Don't delete or edit the following line:
       print(f"In the 2021 Season, Cal Football won {cal_wins} games and lost␣
         ↪{cal_losses} games. Go Bears!  ")
```

```
[True, True, False, True, True, True, False, False, True, False, True, False]
Opponent          | Cal Score | Opponent Score | Results
Nevada            | 17        | 22             | True
```

```
TCU               | 32        | 34              | True
Sacramento State  | 42        | 30              | False
Washington        | 24        | 31              | True
Washington State  | 6         | 21              | True
Oregon            | 17        | 24              | True
Colorado          | 26        | 3               | False
Oregon State      | 39        | 25              | False
Arizona           | 3         | 10              | True
Stanford          | 41        | 11              | False
… (2 rows omitted)
7
5
In the 2021 Season, Cal Football won 5 games and lost 7 games. Go Bears!
```

[128]:
```
grader.check("q1_4")
```

[128]:
```
q1_4 results: All test cases passed!
```

Question 5: Sometimes in football the two teams are equally matched and the game is quite close. Other times, it is a blowout, where the winning team wins by a large margin of victory. Let's define a **big win** to be a game in which the winning team won by more than 10 points.

Use your `final_scores` table to assign `big_wins` to an array of team names that Cal had big wins against during the 2021 football season. You may find the `is_big_win` function defined below helpful to you! **(20 Points)**

[149]:
```
def is_big_win(row):
    '''Return a boolean to describe whether or not a game (row) is a big win'''
    score_diff = row.item("Cal Score") - row.item("Opponent Score")

    if score_diff > 10:
        return True
    else:
        return False

big_wins = make_array()

for row in final_scores.rows: # This will help us iterate through rows of␣
 ↪final_scores table
    opponent = row.item(2)
    print(f'Opponent score: {opponent}. \t Cal score: {row.item(1)}')
    if is_big_win(row): # You should use the function defined above!
        big_wins = np.append(big_wins, opponent) # Do not change this line -␣
 ↪Adds character to the end of some_string
big_wins
```

```
Opponent score: 22.      Cal score: 17
Opponent score: 34.      Cal score: 32
Opponent score: 30.      Cal score: 42
Opponent score: 31.      Cal score: 24
Opponent score: 21.      Cal score: 6
Opponent score: 24.      Cal score: 17
Opponent score: 3.       Cal score: 26
Opponent score: 25.      Cal score: 39
Opponent score: 10.      Cal score: 3
Opponent score: 11.      Cal score: 41
Opponent score: 42.      Cal score: 14
Opponent score: 14.      Cal score: 24
```

```
[149]: array([ 30.,   3.,  25.,  11.])
```

```
[150]: grader.check("q1_5")
```

```
[150]: q1_5 results: All test cases passed!
```

You're done with the required section of Homework 5! Continue on to the optional section for some more practice with iterations and for loops.

Make sure to **run the submit cell located at the bottom of this notebook.**

1.2 2. Unrolling Loops (Optional)

This section of HW5 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 a `for` loop.

In the 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!)

Optional Question 1. Unroll the code below.

```
[151]: for joke_iteration in np.arange(3):
           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?
Orange.
Orange who?
Orange you glad I didn't say banana?
```

[ ]:
```
...
```

Congratulations, you are done with Homework 5!

Important submission steps: 1. Run the tests and verify that they all pass. 2. Choose **Save Notebook** from the **File** menu, then **run the final cell**. 3. Choose **PDF via LaTeX(.pdf)** at **Download as** from the **File** menu. 4. Then submit the PDF file to the corresponding assignment according to your instructor's directions.

It is your responsibility to make sure your work is saved before running the last cell.

1.3 Submission

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output. The cell below will generate a zip file for you to submit. **Please save before exporting!**

[152]:
```
# Save your notebook first, then run this cell to export your submission.
grader.export(pdf=False, run_tests=True)
```

```
Running your submission against local test cases…
```

```
Your submission received the following results when run against available test
cases:

    q1_1 results: All test cases passed!

    q1_2 results: All test cases passed!

    q1_3 results: All test cases passed!

    q1_4 results: All test cases passed!

    q1_5 results: All test cases passed!
<IPython.core.display.HTML object>
```