In [ ]:
```
# Initialize Otter
import otter
grader = otter.Notebook("hw06.ipynb")
```

1 Homework 6: Iteration and Chance

1.1 References

- Sections 9.0 - 9.3
- Chapter 10
- `datascience` Documentation
- Python Quick Reference

1.2 Assignment Reminders

- Make sure to run the code cell at the top of this notebook that starts with `# Initialize Otter` to load the auto-grader.
- For all tasks indicated with a that you must write explanations and sentences for, provide your answer in the designated space.
- Throughout this assignment 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!*
- We encourage you to discuss this assignment with others but make sure to write and submit your own code. Refer to the syllabus to learn more about how to learn cooperatively.
- Unless you are asked otherwise, use the non-interactive visualizations when asked to produce a visualization for a task.
- View the related Canvas Assignment page for additional details.

Run the following code cell to import the tools for this assignment.

In [1]:
```
from datascience import *
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
```

1.3 2022-23 CCSF Football Season

You are going to analyze how well the CCSF football team performed in the 2022-23 season. A football game is divided into four periods, called quarters. The number of points CCSF scored in each quarter, and

the number of points their opponent scored in each quarter are stored in a table called `ccsf_fb.csv`.

Note that the 2022-23 season data was collected from the CCSF Athletics website.

```
In [2]: games = Table().read_table("ccsf_fb.csv")
        games.show()
```

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

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

1.3.1 Task 01

Write a function called `sum_scores`. It should take five arguments, where each argument is the team's quarter or overt time score. It should return the team's total score for that game.

Points: 2

```
In [3]: ''' # BEGIN PROMPT
        def sum_scores(...):
            """Returns the total score calculated by adding up the score of each quarter and the over t
            ...
        '''; # END PROMPT
        # BEGIN SOLUTION NO PROMPT
        def sum_scores(q1, q2, q3, q4, ot):
            """Returns the total score calculated by adding up the score of each quarter and the over t
            return q1 + q2 + q3 + q4 + ot
        # END SOLUTION

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

```
Out[3]: 27
```

```
In [ ]: grader.check("task_01")
```

1.3.2 Task 02

Create a new table `final_scores` with three columns in this *specific* order: `Opponent`, `CCSF Score`, `Opponent Score`. You will have to create the `CCSF 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 `ccsf_scores` and `opponent_scores` output what you want.

Points: 3

```
In [6]: ccsf_scores = games.apply(sum_scores, "CCSF 1Q", "CCSF 2Q", "CCSF 3Q", "CCSF 4Q", "CCSF OT") # A
        opponent_scores = games.apply(sum_scores, "Opp 1Q", "Opp 2Q", "Opp 3Q", "Opp 4Q", "Opp OT") # Su
        final_scores = games.with_columns('CCSF Score', ccsf_scores, "Opponent Score", opponent_scores)
        final_scores
```

```
Out[6]: Opponent          | CCSF Score | Opponent Score
        Santa Rosa        | 31         | 20
        Sacramento City   | 30         | 10
        Butte             | 14         | 35
        Fresno            | 24         | 6
        Sierra            | 10         | 14
        San Joaquin Delta | 44         | 17
        Laney             | 20         | 31
        Diablo Valley     | 17         | 16
        Chabot            | 38         | 14
        San Mateo         | 13         | 31
        … (1 rows omitted)
```

```
In [ ]: grader.check("task_02")
```

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
         games.row(10)
```

```
Out[10]: Row(Opponent='Sequoias', CCSF 1Q=10, CCSF 2Q=14, CCSF 3Q=0, CCSF 4Q=10, CCSF OT=0, Opp 1Q=7, Op
```

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

```
Out[11]: 10
```

1.3.3 Task 03

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

Note 1: "Row object" means a row from the table extracted (behind the scenes) using `tbl.row(index)` 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!

Points: 3

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

```
In [ ]: grader.check("task_03")
```

1.3.4 Task 04

Unfortunately, CCSF did not win against every opponent during the 2022-23 season. Using the `final_scores` table, assign `results` to an array of `True` and `False` values that correspond to whether or not CCSF 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 CCSF had to `ccsf_wins` and `ccsf_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`

Points: 4

```
In [16]: results = final_scores.apply(did_ccsf_win) # SOLUTION
         final_scores_with_results = final_scores.with_columns("Results", results) # SOLUTION
```

```
ccsf_wins = final_scores_with_results.where("Results", True).num_rows # SOLUTION
ccsf_losses = final_scores_with_results.num_rows - ccsf_wins # SOLUTION

# Don't delete or edit the following line:
print(f"In the 2022-23 Season, CCSF Football won {ccsf_wins} games and lost {ccsf_losses} game
```

```
In the 2022-23 Season, CCSF Football won 7 games and lost 4 games. Go RAMS!
```

```
In [ ]: grader.check("task_04")
```

1.3.5 Task 05

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.

Create a function called `is_big_win`. * The function should accept a single row (`datascience.tables.Row` data type) from a table like the `final_scores` table. * The function should output `True` if CCSF won by **more than** 10 points. Otherwise, it should return `False`.

Test your function on the first row of the table. `final_scores.row(0)` should be the row object `Row(Opponent='Santa Rosa', CCSF Score=31, Opponent Score=20)`. Since CCSF's score is more than 10 points larger than Santa Rosa's score, then `is_big_win` applied to that row should output `True`.

Points: 4

```
In [21]: """ # BEGIN PROMPT
         ...

         # Test your function
         row1 = final_scores.row(0)
         is_big_win(row1)
         """; # END PROMPT
         # BEGIN SOLUTION NO PROMPT
         def is_big_win(a_row):
             '''Return a boolean to describe whether or not a game (row) is a big win'''
             score_diff = a_row.item("CCSF Score") - a_row.item("Opponent Score")

             if score_diff > 10:
                 return True
             else:
                 return False

         # Test your function
```

```
row1 = final_scores.row(0)
is_big_win(row1)
# END SOLUTION
```

Out[21]: True

In [ ]:
```
grader.check("task_05")
```

1.3.6 Task 06

Use your `final_scores` table with your `is_big_win` function to assign `big_wins` to an array of team names that CCSF had big wins against during the 2022-23 football season.

Points: 5

In [28]:
```
""" # BEGIN PROMPT
big_wins = ...

for row_index ...
    row = ...
    opponent = ...
    if ...:
        big_wins = np.append(big_wins, ...)
"""; # END PROMPT
# BEGIN SOLUTION NO PROMPT
big_wins = make_array()

for row_index in np.arange(final_scores.num_rows):
    row = final_scores.row(row_index)
    opponent = row.item("Opponent")
    if is_big_win(row):
        big_wins = np.append(big_wins, opponent)
# END SOLUTION

big_wins
```

Out[28]:
```
array(['Santa Rosa', 'Sacramento City', 'Fresno', 'San Joaquin Delta',
       'Chabot'],
      dtype='<U32')
```

In [ ]:
```
grader.check("task_06")
```

1.4 Roulette

A Nevada roulette wheel has 38 pockets and a small ball that rests on the wheel. When the wheel is spun, the ball comes to rest in one of the 38 pockets. That pocket is declared the winner.

The pockets are labeled 0, 00, 1, 2, 3, 4, … , 36. Pockets 0 and 00 are green, and the other pockets are alternately red and black. The table `wheel` is a representation of a Nevada roulette wheel. **Note that *both* columns consist of strings.** Below is an example of a roulette wheel!

1.4.1 Chance

Before you do the following tasks, make sure you understand the logic behind all the examples in Section 9.5.

Good ways to approach probability calculations include:

- Thinking one trial at a time: What does the first one have to be? Then what does the next one have to be?
- Breaking up the event into distinct ways in which it can happen.
- Seeing if it is easier to find the chance that the event does not happen.

On each spin of a roulette wheel, all 38 pockets are equally likely to be the winner regardless of the results of other spins. Among the 38 pockets, 18 are red, 18 black, and 2 green.

Task 07 The winning pocket is black on all of the first three spins.

Provide an expression that Python evaluates to the chance of the event described.

Points: 2

```
In [34]: first_three_black = (18 / 38) ** 3# SOLUTION
```

```
In [ ]: grader.check("task_07")
```

Task 08 The color green never wins in the first 10 spins.

Provide an expression that Python evaluates to the chance of the event described.

Points: 2

```
In [37]: no_green = (1 - (2 / 38)) ** 10# SOLUTION
```

```
In [ ]: grader.check("task_08")
```

Task 09 The color green wins **at least** once on the first 10 spins.

Provide an expression that Python evaluates to the chance of the event described.

Points: 2

```
In [40]: at_least_one_green = 1 - (1 - 2 / 38) ** 10# SOLUTION
```

```
In [ ]: grader.check("task_09")
```

Task 10 One of the three colors **never** wins in the first 10 spins.

Provide an expression that Python evaluates to the chance of the event described.

Points: 2

```
In [43]: lone_winners = (1 - (2 / 38)) ** 10 + 2 * (1 - (18 / 38)) ** 10 # SOLUTION
```

```
In [ ]: grader.check("task_10")
```

1.4.2 Comparing Chances

In each of the following 3 questions, two events A and B are described. Choose from one of the following three options and set each answer variable to a single integer:

1. Event A is more likely than Event B
2. Event B is more likely than Event A
3. The two events have the same chance.

You should be able to make the choices **without calculation**. Good ways to approach this exercise include imagining carrying out the chance experiments yourself, one trial at a time, and by thinking about the law of averages.

Task 11 A child picks four times at random from a box that has four toy animals: a bear, an elephant, a giraffe, and a kangaroo.

- Event A: all four different animals are picked, assuming the child picks without replacement
- Event B: all four different animals are picked, assuming the child picks with replacement

Points: 2

```
In [46]: toys_option = 1 # SOLUTION
```

```
In [ ]: grader.check("task_11")
```

Task 12 In a lottery, two numbers are drawn at random without replacement from the integers 1 through 1000.

- Event A: The number 8 is picked on both draws
- Event B: The same number is picked on both draws

Points: 2

```
In [49]: lottery_option = 3 # SOLUTION
```

```
In [ ]: grader.check("task_12")
```

1.5 Submit your Homework to Canvas

Once you have finished working on the homework tasks, prepare to submit your work in Canvas by completing the following steps.

1. In the related Canvas Assignment page, check the rubric to know how you will be scored for this assignment.

2. Double-check that you have run the code cell near the end of the notebook that contains the command `"grader.check_all()"`. This command will run all of the run tests on all your responses to the auto-graded tasks marked with .
3. Double-check your responses to the manually graded tasks marked with .
4. Select the menu item "File" and "Save Notebook" in the notebook's Toolbar to save your work and create a specific checkpoint in the notebook's work history.
5. Select the menu items "File", "Download" in the notebook's Toolbar to download the notebook (.ipynb) file.
6. In the related Canvas Assignment page, click Start Assignment or New Attempt to upload the downloaded .ipynb file.

Keep in mind that the autograder does not always check for correctness. Sometimes it just checks for the format of your answer, so passing the autograder for a question does not mean you got the answer correct for that question.

To double-check your work, the cell below will rerun all of the autograder tests.

In [ ]:
```
grader.check_all()
```