hw02

March 10, 2022

Name: Allan Gongora

Section: 0131

1 Homework 2: Arrays and Tables

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

[1]:
```
pip install gofer-grader
```

```
Requirement already satisfied: gofer-grader in /opt/conda/lib/python3.7/site-
packages (1.1.0)
Requirement already satisfied: pygments in /opt/conda/lib/python3.7/site-
packages (from gofer-grader) (2.11.2)
Requirement already satisfied: jinja2 in /opt/conda/lib/python3.7/site-packages
(from gofer-grader) (3.0.3)
Requirement already satisfied: tornado in /opt/conda/lib/python3.7/site-packages
(from gofer-grader) (6.1)
Requirement already satisfied: MarkupSafe>=2.0 in /opt/conda/lib/python3.7/site-
packages (from jinja2->gofer-grader) (2.0.1)
Note: you may need to restart the kernel to use updated packages.
```

[2]:
```
# Don't change this cell; just run it.

import numpy as np
from datascience import *

from gofer.ok import check
```

Recommended Reading: * Data Types * Sequences * Tables

1) For all problems that you must write explanations and sentences for, you **must** provide your answer in the designated space. This can include:
   A) Sentence reponses to questions that ask for an explanation
   B) Numeric responses to multiple choice questions
   C) Programming code
2) 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!

Once you're finished, select "Save and Checkpoint" in the File menu. Your name and course section number should be in the first and last cell of the assignment. Be sure you have run all cells with code and that the output from that is showing. Then click "Print Preview" in the File menu. Print a copy from there in pdf format. (This means you right click and choose print and choose "save as pdf" from your printer options.) You will need to submit the pdf in Canvas by the deadline.

The gopher grader output and/or output from your coding are essential to helping your instructor grade your work correctly and in a timely manner.

Files submitted that are missing the required output will lose some to all points so double check your pdf before submitting.

1.1 1. Creating Arrays

Question 1.1 Make an array called `weird_numbers` containing the following numbers (in the given order):

1. -2
2. the sine of 1.2
3. 3
4. 5 to the power of the cosine of 1.2

Hint: `sin` and `cos` are functions in the `math` module.

[3]:
```
# Our solution involved one extra line of code before creating
# weird_numbers.
weird_numbers = make_array(-2,np.sin(1.2), 3, 5**np.cos(1.2))
weird_numbers
```

[3]:
```
array([-2.        ,  0.93203909,  3.        ,  1.79174913])
```

[4]:
```
check('tests/q1_1.py')
```

[4]:
```
<gofer.ok.OKTestsResult at 0x7f156f85b4d0>
```

Question 1.2 Make an array called `book_title_words` containing the following three strings: "Eats", "Shoots", and "and Leaves".

[5]:
```
book_title_words = make_array("Eats", "Shoots", "and Leaves")
book_title_words
```

[5]:
```
array(['Eats', 'Shoots', 'and Leaves'], dtype='<U10')
```

[6]:
```
check('tests/q1_2.py')
```

[6]:
```
<gofer.ok.OKTestsResult at 0x7f1569320c90>
```

Strings have a method called `join`. `join` takes one argument, an array of strings. It returns a single string. Specifically, the value of `a_string.join(an_array)` is a single string that's the concatenation ("putting together") of all the strings in `an_array`, **except** `a_string` is inserted in between each string.

Question 1.3 Use the array `book_title_words` and the method `join` to make two strings:

1. "Eats, Shoots, and Leaves" (call this one `with_commas`)
2. "Eats Shoots and Leaves" (call this one `without_commas`)

Hint: If you're not sure what `join` does, first try just calling, for example, `"foo".join(book_title_words)` .

[7]:
```
with_commas = ", ".join(book_title_words)
without_commas = " ".join(book_title_words)

# These lines are provided just to print out your answers.
print('with_commas:', with_commas)
print('without_commas:', without_commas)
```

```
with_commas: Eats, Shoots, and Leaves
without_commas: Eats Shoots and Leaves
```

[8]:
```
check('tests/q1_3.py')
```

[8]:
```
<gofer.ok.OKTestsResult at 0x7f156935ef50>
```

1.2 2. Indexing Arrays

These exercises give you practice accessing individual elements of arrays. In Python (and in many programming languages), elements are accessed by *index*, so the first element is the element at index 0.

Question 2.1 The cell below creates an array of some numbers. Set `third_element` to the third element of `some_numbers`.

[9]:
```
some_numbers = make_array(-1, -3, -6, -10, -15)

third_element = some_numbers[2]
third_element
```

[9]:
```
-6
```

[10]:
```
check('tests/q2_1.py')
```

[10]:
```
<gofer.ok.OKTestsResult at 0x7f1569236d50>
```

Question 2.2 The next cell creates a table that displays some information about the elements of `some_numbers` and their order. Run the cell to see the partially-completed table, then fill in the missing information in the cell (the strings that are currently "???") to complete the table.

[11]:
```
elements_of_some_numbers = Table().with_columns(
    "English name for position", make_array("first", "second", "third",␣
 ↪"fourth", "fifth"),
    "Index",                     make_array("0", "1", "2", "3", "4"),
    "Element",                   some_numbers)
elements_of_some_numbers
```

[11]:

English name for position	Index	Element
first	0	-1
second	1	-3
third	2	-6
fourth	3	-10
fifth	4	-15

[12]:
```
check('tests/q2_2.py')
```

[12]:
```
<gofer.ok.OKTestsResult at 0x7f156935ee50>
```

Question 2.3 You'll sometimes want to find the *last* element of an array. Suppose an array has 142 elements. What is the index of its last element?

[13]:
```
index_of_last_element = 141
```

[14]:
```
check('tests/q2_3.py')
```

[14]:
```
<gofer.ok.OKTestsResult at 0x7f1569350750>
```

More often, you don't know the number of elements in an array, its *length*. (For example, it might be a large dataset you found on the Internet.) The function `len` takes a single argument, an array, and returns the `len`gth of that array (an integer).

Question 2.4 The cell below loads an array called `president_birth_years`. The last element in that array is the most recent birth year of any deceased president. Assign that year to `most_recent_birth_year`.

[15]:
```
president_birth_years = Table.read_table("president_births.csv").column('Birth␣
 ↪Year')

most_recent_birth_year = president_birth_years[-1]
most_recent_birth_year
```

[15]:
```
1917
```

[16]:
```
check('tests/q2_4.py')
```

[16]:
```
<gofer.ok.OKTestsResult at 0x7f1569320a50>
```

Question 2.5 Finally, assign `sum_of_birth_years` to the sum of the first, tenth, and last birth year in `president_birth_years`

[17]:
```
sum_of_birth_years = sum([president_birth_years[0], president_birth_years[9],␣
↪president_birth_years[-1]])
```

[18]:
```
check('tests/q2_5.py')
```

[18]:
```
<gofer.ok.OKTestsResult at 0x7f1569380e10>
```

1.3 3. Basic Array Arithmetic

Question 3.1 Multiply the numbers 42, 4224, 42422424, and -250 by 157. For this question, **don't** use arrays.

[19]:
```
first_product, second_product, third_product, fourth_product = [i * 157 for i␣
↪in [42, 4224, 42422424, -250]]

print(first_product, second_product, third_product, fourth_product)
```

```
6594 663168 6660320568 -39250
```

[20]:
```
check('tests/q3_1.py')
```

[20]:
```
<gofer.ok.OKTestsResult at 0x7f156924a0d0>
```

Question 3.2 Now, do the same calculation, but using an array called `numbers` and only a single multiplication (`*`) operator. Store the 4 results in an array named `products`.

[21]:
```
numbers = make_array(42, 4224, 42422424, -250)
products = numbers * 157
products
```

[21]:
```
array([      6594,     663168, 6660320568,     -39250])
```

[22]:
```
check('tests/q3_2.py')
```

[22]:
```
<gofer.ok.OKTestsResult at 0x7f1569256b10>
```

Question 3.3 Oops, we made a typo! Instead of 157, we wanted to multiply each number by 1577. Compute the fixed products in the cell below using array arithmetic. Notice that your job is really easy if you previously defined an array containing the 4 numbers.

[23]:
```
fixed_products = numbers * 1577
fixed_products
```

[23]:
```
array([      66234,     6661248, 66900162648,     -394250])
```

[24]:
```
check('tests/q3_3.py')
```

[24]:
```
<gofer.ok.OKTestsResult at 0x7f15691ef910>
```

Question 3.4 We've loaded an array of temperatures in the next cell. Each number is the highest temperature observed on a day at a climate observation station, mostly from the US. Since they're from the US government agency NOAA, all the temperatures are in Fahrenheit. Convert them all to Celsius by first subtracting 32 from them, then multiplying the results by $\frac{5}{9}$. Make sure to **ROUND** each result to the nearest integer using the `np.round` function.

[25]:
```
max_temperatures = Table.read_table("temperatures.csv").column("Daily Max␣
↪Temperature")

celsius_max_temperatures = np.round((max_temperatures - 32) * (5/9))
celsius_max_temperatures
```

[25]:
```
array([-4., 31., 32., …, 17., 23., 16.])
```

[26]:
```
check('tests/q3_4.py')
```

[26]:
```
<gofer.ok.OKTestsResult at 0x7f15691ef890>
```

Question 3.5 The cell below loads all the *lowest* temperatures from each day (in Fahrenheit). Compute the size of the daily temperature range for each day. That is, compute the difference between each daily maximum temperature and the corresponding daily minimum temperature. **Give your answer in Celsius!** Make sure **NOT** to round your answer for this question!

[27]:
```
min_temperatures = Table.read_table("temperatures.csv").column("Daily Min␣
↪Temperature")

celsius_temperature_ranges = (max_temperatures - 32) * (5/9) -␣
↪(min_temperatures - 32) * (5/9)
celsius_temperature_ranges
```

[27]:
```
array([ 6.66666667, 10.        , 12.22222222, …, 17.22222222,
       11.66666667, 11.11111111])
```

[28]:
```
check('tests/q3_5.py')
```

[28]:
```
<gofer.ok.OKTestsResult at 0x7f1569382f50>
```

1.4 4. World Population

The cell below loads a table of estimates of the world population for different years, starting in 1950. The estimates come from the US Census Bureau website.

[29]:
```
world = Table.read_table("world_population.csv").select('Year', 'Population')
world.show(4)
```

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

The name `population` is assigned to an array of population estimates.

[30]:
```
population = world.column(1)
population
```

[30]:
```
array([2557628654, 2594939877, 2636772306, 2682053389, 2730228104,
       2782098943, 2835299673, 2891349717, 2948137248, 3000716593,
       3043001508, 3083966929, 3140093217, 3209827882, 3281201306,
       3350425793, 3420677923, 3490333715, 3562313822, 3637159050,
       3712697742, 3790326948, 3866568653, 3942096442, 4016608813,
       4089083233, 4160185010, 4232084578, 4304105753, 4379013942,
       4451362735, 4534410125, 4614566561, 4695736743, 4774569391,
       4856462699, 4940571232, 5027200492, 5114557167, 5201440110,
       5288955934, 5371585922, 5456136278, 5538268316, 5618682132,
       5699202985, 5779440593, 5857972543, 5935213248, 6012074922,
       6088571383, 6165219247, 6242016348, 6318590956, 6395699509,
       6473044732, 6551263534, 6629913759, 6709049780, 6788214394,
       6866332358, 6944055583, 7022349283, 7101027895, 7178722893,
       7256490011])
```

In this question, you will apply some built-in Numpy functions to this array.

The difference function `np.diff` subtracts each element in an array by the element that preceeds it. As a result, the length of the array `np.diff` returns will always be one less than the length of the input array.

The cumulative sum function `np.cumsum` outputs an array of partial sums. For example, the third element in the output array corresponds to the sum of the first, second, and third elements.

Question 4.1 Very often in data science, we are interested understanding how values change with time. Use `np.diff` and `np.max` (or just `max`) to calculate the largest annual change in population between any two consecutive years.

[31]:
```
largest_population_change = max(np.diff(population))
largest_population_change
```

[31]:
```
87515824
```

[32]:
```
check('tests/q4_1.py')
```

[32]:
```
<gofer.ok.OKTestsResult at 0x7f15691fd7d0>
```

Question 4.2 Describe in words the result of the following expression. What do the values in the resulting array represent (choose one)?

[33]:
```
np.cumsum(np.diff(population))
```

[33]:
```
array([  37311223,   79143652,  124424735,  172599450,  224470289,
        277671019,  333721063,  390508594,  443087939,  485372854,
        526338275,  582464563,  652199228,  723572652,  792797139,
        863049269,  932705061, 1004685168, 1079530396, 1155069088,
```

```
1232698294, 1308939999, 1384467788, 1458980159, 1531454579,
1602556356, 1674455924, 1746477099, 1821385288, 1893734081,
1976781471, 2056937907, 2138108089, 2216940737, 2298834045,
2382942578, 2469571838, 2556928513, 2643811456, 2731327280,
2813957268, 2898507624, 2980639662, 3061053478, 3141574331,
3221811939, 3300343889, 3377584594, 3454446268, 3530942729,
3607590593, 3684387694, 3760962302, 3838070855, 3915416078,
3993634880, 4072285105, 4151421126, 4230585740, 4308703704,
4386426929, 4464720629, 4543399241, 4621094239, 4698861357])
```

1) The total population change between consecutive years, starting at 1951.

2) The total population change between 1950 and each later year, starting at 1951.

3) The total population change between 1950 and each later year, starting inclusively at 1950.

[34]:
```
# Assign cumulative_sum_answer to 1, 2, or 3
cumulative_sum_answer = 3
```

[35]:
```
check('tests/q4_2.py')
```

[35]:
```
<gofer.ok.OKTestsResult at 0x7f15691fd150>
```

1.5 5. Old Faithful

Important: In this question, the `gofer` tests don't tell you whether or not your answer is correct. They only check that your answer is close. However, when the question is graded, we will check for the correct answer. Therefore, you should do your best to submit answers that not only pass the tests, but are also correct.

Old Faithful is a geyser in Yellowstone that erupts every 44 to 125 minutes (according to Wikipedia). People are often told that the geyser erupts every hour, but in fact the waiting time between eruptions is more variable. Let's take a look.

Question 5.1 The first line below assigns `waiting_times` to an array of 272 consecutive waiting times between eruptions, taken from a classic 1938 dataset. Assign the names `shortest`, `longest`, and `average` so that the `print` statement is correct.

[36]:
```
waiting_times = Table.read_table('old_faithful.csv').column('waiting')

shortest = waiting_times.min()
longest = waiting_times.max()
average = waiting_times.mean()

print("Old Faithful erupts every", shortest, "to", longest, "minutes and␣
  ↪every", average, "minutes on average.")
```

```
Old Faithful erupts every 43 to 96 minutes and every 70.8970588235294 minutes on
average.
```

[37]:
```
check('tests/q5_1.py')
```

[37]:
```
<gofer.ok.OKTestsResult at 0x7f15691fd910>
```

Question 5.2 Assign `biggest_decrease` to the biggest decrease in waiting time between two consecutive eruptions. For example, the third eruption occurred after 74 minutes and the fourth after 62 minutes, so the decrease in waiting time was 74 - 62 = 12 minutes.

Hint: You'll need an array arithmetic function mentioned in the textbook.

Hint 2: The biggest decrease could be negative, but in the end, we want to return the absolute value of the biggest decrease

[38]:
```
biggest_decrease = abs(np.diff(waiting_times).min())
biggest_decrease
```

[38]:
```
45
```

[39]:
```
check('tests/q5_2.py') # this isn't working. im confident my solution is correct
```

[39]:
```
<gofer.ok.OKTestsResult at 0x7f15b05756d0>
```

Question 5.3 If you expected Old Faithful to erupt every hour, you would expect to wait a total of `60 * k` minutes to see `k` eruptions. Set `difference_from_expected` to an array with 272 elements, where the element at index `i` is the absolute difference between the expected and actual total amount of waiting time to see the first `i+1` eruptions.

Hint: You'll need to compare a cumulative sum to a range.

For example, since the first three waiting times are 79, 54, and 74, the total waiting time for 3 eruptions is 79 + 54 + 74 = 207. The expected waiting time for 3 eruptions is 60 * 3 = 180. Therefore, `difference_from_expected.item(2)` should be $|207 - 180| = 27$.

[40]:
```
difference_from_expected = abs(np.cumsum(waiting_times) - (np.arange(1, 273) *␣
↪60))
difference_from_expected
```

[40]:
```
array([  19,   13,   27,   29,   54,   49,   77,  102,   93,  118,  112,
        136,  154,  141,  164,  156,  158,  182,  174,  193,  184,  171,
        189,  198,  212,  235,  230,  246,  264,  283,  296,  313,  319,
        339,  353,  345,  333,  353,  352,  382,  402,  400,  424,  422,
        435,  458,  462,  455,  477,  476,  491,  521,  515,  535,  529,
        552,  563,  567,  584,  605,  604,  628,  616,  638,  638,  670,
        688,  706,  711,  724,  746,  742,  761,  772,  774,  790,  790,
        808,  824,  847,  862,  884,  894,  899,  912,  940,  956,  976,
        964,  990,  990, 1020, 1010, 1028, 1031, 1043, 1067, 1082, 1073,
       1095, 1097, 1125, 1114, 1137, 1158, 1145, 1169, 1161, 1187, 1208,
       1223, 1222, 1251, 1270, 1269, 1290, 1280, 1305, 1304, 1331, 1324,
       1333, 1350, 1346, 1374, 1395, 1380, 1402, 1397, 1427, 1412, 1435,
       1431, 1460, 1446, 1468, 1459, 1485, 1478, 1497, 1518, 1518, 1540,
```

```
       1557, 1573, 1572, 1592, 1581, 1617, 1610, 1627, 1644, 1649, 1670,
       1681, 1691, 1712, 1745, 1738, 1767, 1752, 1778, 1776, 1794, 1800,
       1816, 1819, 1847, 1839, 1872, 1861, 1858, 1875, 1883, 1904, 1925,
       1938, 1928, 1953, 1967, 1962, 1979, 2002, 2025, 2016, 2034, 2058,
       2044, 2067, 2062, 2083, 2080, 2096, 2120, 2137, 2158, 2185, 2202,
       2193, 2211, 2211, 2233, 2264, 2257, 2275, 2261, 2278, 2302, 2291,
       2314, 2325, 2345, 2334, 2349, 2353, 2369, 2362, 2396, 2391, 2407,
       2397, 2419, 2413, 2428, 2446, 2465, 2483, 2501, 2511, 2530, 2540,
       2534, 2560, 2550, 2580, 2574, 2568, 2585, 2604, 2608, 2623, 2610,
       2636, 2639, 2664, 2686, 2683, 2705, 2712, 2726, 2720, 2743, 2756,
       2769, 2797, 2817, 2828, 2851, 2847, 2866, 2884, 2908, 2906, 2929,
       2912, 2912, 2927, 2948, 2934, 2964, 2950, 2964])
```

[41]:
```
check('tests/q5_3.py')
```

[41]:
```
<gofer.ok.OKTestsResult at 0x7f15691fb990>
```

Question 5.4 If instead you guess that each waiting time will be the same as the previous waiting time, how many minutes would your guess differ from the actual time, averaging over every wait time except the first one.

For example, since the first three waiting times are 79, 54, and 74, the average difference between your guess and the actual time for just the second and third eruption would be $\frac{|79-54|+|54-74|}{2} = 22.5$.

[42]:
```
waiting_times
```

[42]:
```
array([79, 54, 74, 62, 85, 55, 88, 85, 51, 85, 54, 84, 78, 47, 83, 52, 62,
       84, 52, 79, 51, 47, 78, 69, 74, 83, 55, 76, 78, 79, 73, 77, 66, 80,
       74, 52, 48, 80, 59, 90, 80, 58, 84, 58, 73, 83, 64, 53, 82, 59, 75,
       90, 54, 80, 54, 83, 71, 64, 77, 81, 59, 84, 48, 82, 60, 92, 78, 78,
       65, 73, 82, 56, 79, 71, 62, 76, 60, 78, 76, 83, 75, 82, 70, 65, 73,
       88, 76, 80, 48, 86, 60, 90, 50, 78, 63, 72, 84, 75, 51, 82, 62, 88,
       49, 83, 81, 47, 84, 52, 86, 81, 75, 59, 89, 79, 59, 81, 50, 85, 59,
       87, 53, 69, 77, 56, 88, 81, 45, 82, 55, 90, 45, 83, 56, 89, 46, 82,
       51, 86, 53, 79, 81, 60, 82, 77, 76, 59, 80, 49, 96, 53, 77, 77, 65,
       81, 71, 70, 81, 93, 53, 89, 45, 86, 58, 78, 66, 76, 63, 88, 52, 93,
       49, 57, 77, 68, 81, 81, 73, 50, 85, 74, 55, 77, 83, 83, 51, 78, 84,
       46, 83, 55, 81, 57, 76, 84, 77, 81, 87, 77, 51, 78, 60, 82, 91, 53,
       78, 46, 77, 84, 49, 83, 71, 80, 49, 75, 64, 76, 53, 94, 55, 76, 50,
       82, 54, 75, 78, 79, 78, 78, 70, 79, 70, 54, 86, 50, 90, 54, 54, 77,
       79, 64, 75, 47, 86, 63, 85, 82, 57, 82, 67, 74, 54, 83, 73, 73, 88,
       80, 71, 83, 56, 79, 78, 84, 58, 83, 43, 60, 75, 81, 46, 90, 46, 74])
```

[43]:
```
average_error = abs(np.diff(waiting_times)).sum() / (len(waiting_times) - 1)
average_error
```

[43]:
```
20.52029520295203
```

[44]:
```
check('tests/q5_4.py')
```

[44]:
```
<gofer.ok.OKTestsResult at 0x7f156935e9d0>
```

1.6 6. Tables

Question 6.1 Suppose you have 4 apples, 3 oranges, and 3 pineapples. (Perhaps you're using Python to solve a high school Algebra problem.) Create a table that contains this information. It should have two columns: "fruit name" and "count". Give it the name `fruits`.

Note: Use lower-case and singular words for the name of each fruit, like `"apple"`.

[45]:
```
# Our solution uses 1 statement split over 3 lines.
fruits = Table().with_columns(
    [
        "fruit name", ["apple", "orange", "pineapple"],
        "count", [4, 3, 3]
    ]
)

fruits
```

[45]:
```
fruit name | count
apple      | 4
orange     | 3
pineapple  | 3
```

[46]:
```
check('tests/q6_1.py')
```

[46]:
```
<gofer.ok.OKTestsResult at 0x7f1569220f90>
```

Question 6.2 The file `inventory.csv` contains information about the inventory at a fruit stand. Each row represents the contents of one box of fruit. Load it as a table named `inventory`.

[47]:
```
inventory = Table.read_table("inventory.csv")
inventory
```

[47]:
```
box ID | fruit name | count
53686  | kiwi       | 45
57181  | strawberry | 123
25274  | apple      | 20
48800  | orange     | 35
26187  | strawberry | 255
57930  | grape      | 517
52357  | strawberry | 102
43566  | peach      | 40
```

[48]:
```
check('tests/q6_2.py')
```

[48]:
```
<gofer.ok.OKTestsResult at 0x7f1569220810>
```

Question 6.3 Does each box at the fruit stand contain a different fruit?

[49]:
```
# Set all_different to "Yes" if each box contains a different fruit or to "No"␣
↪if multiple boxes contain the same fruit
all_different = "No"
all_different
```

[49]:
```
'No'
```

[50]:
```
check('tests/q6_3.py')
```

[50]:
```
<gofer.ok.OKTestsResult at 0x7f156935ef10>
```

Question 6.4 The file `sales.csv` contains the number of fruit sold from each box last Saturday. It has an extra column called "price per fruit ($)" that's the price *per item of fruit* for fruit in that box. The rows are in the same order as the `inventory` table. Load these data into a table called `sales`.

[51]:
```
sales = Table.read_table("sales.csv")
sales
```

[51]:

box ID	fruit name	count sold	price per fruit ($)
53686	kiwi	3	0.5
57181	strawberry	101	0.2
25274	apple	0	0.8
48800	orange	35	0.6
26187	strawberry	25	0.15
57930	grape	355	0.06
52357	strawberry	102	0.25
43566	peach	17	0.8

[52]:
```
check('tests/q6_4.py')
```

[52]:
```
<gofer.ok.OKTestsResult at 0x7f156920b790>
```

Question 6.5 How many fruits did the store sell in total on that day?

[53]:
```
total_fruits_sold = sales.column("count sold").sum()
total_fruits_sold
```

[53]:
```
638
```

[54]:
```
check('tests/q6_5.py')
```

[54]:
```
<gofer.ok.OKTestsResult at 0x7f15691fd450>
```

Question 6.6 What was the store's total revenue (the total price of all fruits sold) on that day?

Hint: If you're stuck, think first about how you would compute the total revenue from just the grape sales.

[55]:
```
total_revenue = sum(sales.column("count sold") * sales.column("price per fruit ($)"))
total_revenue
```

[55]:
```
106.85
```

[56]:
```
check('tests/q6_6.py')
```

[56]:
```
<gofer.ok.OKTestsResult at 0x7f1569212910>
```

Question 6.7 Make a new table called `remaining_inventory`. It should have the same rows and columns as `inventory`, except that the amount of fruit sold from each box should be subtracted from that box's count, so that the "count" is the amount of fruit remaining after Saturday.

[57]:
```
inventory
```

[57]:
```
box ID | fruit name | count
53686  | kiwi       | 45
57181  | strawberry | 123
25274  | apple      | 20
48800  | orange     | 35
26187  | strawberry | 255
57930  | grape      | 517
52357  | strawberry | 102
43566  | peach      | 40
```

[58]:
```
remaining_inventory = inventory.with_column("count", inventory.column("count") - sales.column("count sold")) # a groupby would
remaining_inventory # feel safer. in the problem id's match 1 to 1 in order but in the real world i wouldn't do this
```

[58]:
```
box ID | fruit name | count
53686  | kiwi       | 42
57181  | strawberry | 22
25274  | apple      | 20
48800  | orange     | 0
26187  | strawberry | 230
57930  | grape      | 162
52357  | strawberry | 0
43566  | peach      | 23
```

[59]:
```
check('tests/q6_7.py')
```

[59]:
```
<gofer.ok.OKTestsResult at 0x7f15691b0850>
```

1.7 7. Submission

Once you're finished, select "Save and Checkpoint" in the File menu. Your name and course section number should be in the first and last cell of the assignment. Be sure you have run all cells with code and that the output from that is showing.

Double check that you have completed all of the free response questions as the auto-grader does NOT check that and YOU are responsible for knowing those questions are there and completing them as part of the grade for this homework. When ready, click "Print Preview" in the File menu. Print a copy from there in pdf format. (This means you right click and choose print and choose "save as pdf" from your printer options.) You will need to submit the pdf in Canvas by the deadline.

The gopher grader output and/or output from your coding are essential to helping your instructor grade your work correctly and in a timely manner.

Files submitted that are missing the required output will lose some to all points so double check your pdf before submitting.

[60]:
```
# For your convenience, you can run this cell to run all the tests at once!
import glob
from gofer.ok import grade_notebook
if not globals().get('__GOFER_GRADER__', False):
    display(grade_notebook('hw02.ipynb', sorted(glob.glob('tests/q*.py'))))
```

```
with_commas: Eats, Shoots, and Leaves
without_commas: Eats Shoots and Leaves
6594 663168 6660320568 -39250
Old Faithful erupts every 43 to 96 minutes and every 70.8970588235294 minutes on
average.
['tests/q1_1.py', 'tests/q1_2.py', 'tests/q1_3.py', 'tests/q2_1.py',
'tests/q2_2.py', 'tests/q2_3.py', 'tests/q2_4.py', 'tests/q2_5.py',
'tests/q3_1.py', 'tests/q3_2.py', 'tests/q3_3.py', 'tests/q3_4.py',
'tests/q3_5.py', 'tests/q4_1.py', 'tests/q4_2.py', 'tests/q5_1.py',
'tests/q5_2.py', 'tests/q5_3.py', 'tests/q5_4.py', 'tests/q6_1.py',
'tests/q6_2.py', 'tests/q6_3.py', 'tests/q6_4.py', 'tests/q6_5.py',
'tests/q6_6.py', 'tests/q6_7.py']
Question 1:

<gofer.ok.OKTestsResult at 0x7f1569184350>

Question 2:

<gofer.ok.OKTestsResult at 0x7f15690f9b90>

Question 3:

<gofer.ok.OKTestsResult at 0x7f15690f9fd0>

Question 4:

<gofer.ok.OKTestsResult at 0x7f15690f9c10>
```

```
Question 5:
<gofer.ok.OKTestsResult at 0x7f1569106450>
Question 6:
<gofer.ok.OKTestsResult at 0x7f1569106510>
Question 7:
<gofer.ok.OKTestsResult at 0x7f15691065d0>
Question 8:
<gofer.ok.OKTestsResult at 0x7f1569106dd0>
Question 9:
<gofer.ok.OKTestsResult at 0x7f15690f9e50>
Question 10:
<gofer.ok.OKTestsResult at 0x7f1569106150>
Question 11:
<gofer.ok.OKTestsResult at 0x7f1569106a50>
Question 12:
<gofer.ok.OKTestsResult at 0x7f1569212110>
Question 13:
<gofer.ok.OKTestsResult at 0x7f15690f92d0>
Question 14:
<gofer.ok.OKTestsResult at 0x7f15691eb910>
Question 15:
<gofer.ok.OKTestsResult at 0x7f156920ba10>
Question 16:
<gofer.ok.OKTestsResult at 0x7f1569106b50>
Question 17:
<gofer.ok.OKTestsResult at 0x7f1569106c50>
Question 18:
<gofer.ok.OKTestsResult at 0x7f1569106590>
Question 19:
<gofer.ok.OKTestsResult at 0x7f1569106090>
Question 20:
<gofer.ok.OKTestsResult at 0x7f1569106950>
```

```
Question 21:
<gofer.ok.OKTestsResult at 0x7f1569106990>
Question 22:
<gofer.ok.OKTestsResult at 0x7f1569106850>
Question 23:
<gofer.ok.OKTestsResult at 0x7f15691069d0>
Question 24:
<gofer.ok.OKTestsResult at 0x7f15691062d0>
Question 25:
<gofer.ok.OKTestsResult at 0x7f15690f9bd0>
Question 26:
<gofer.ok.OKTestsResult at 0x7f1569110490>
1.0
```

Name: Allan Gongora

Section: 0131

[ ]: