hw02

September 14, 2024

[73]:
```
# Initialize Otter
import otter
grader = otter.Notebook("hw02.ipynb")
```

1 Homework 2: Arrays and Tables

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

[74]:
```
# Run this cell and ignore the output. It is checking to see if you ran the␣
↪first cell!
# If you get a NameError, please run the cell at the very TOP of this notebook!

grader
```

[74]:
```
<otter.check.notebook.Notebook at 0x7d966dd7a620>
```

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

Recommended Readings: - Arrays - What is Data Science? - Causality and Experiments - Programming in Python

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!

Important: In this homework, the `grader checks` tests will tell you whether your answer is correct, except for Parts 4 & 5. In future homework assignments, correctness tests will typically not be provided.

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. Creating Arrays

[4]:
```
# Run this cell to set up the notebook, but please don't change it.

import numpy as np
from datascience import *
import warnings
warnings.simplefilter('ignore', FutureWarning)
```

Question 1. Make an array called `weird_numbers` containing the following numbers (in the given order) :

1. -2
2. the floor of 12.6
3. 3
4. 5 to the power of the ceil of 5.3

Hint: `floor` and `ceil` are functions in the `math` module. Importing modules is covered in Lab 2!

Note: Python lists are different/behave differently than NumPy arrays. In SBCC's Data Science course, we use NumPy arrays, so please make an **array**, not a Python list.

[24]:
```
# Our solution involved one extra line of code before creating
# weird_numbers.
import math
...
weird_numbers = make_array(-2, math.floor(12.6), 3, math.pow(5, math.ceil(5.3)))
weird_numbers
```

[24]:
```
array([ -2.00000000e+00,   1.20000000e+01,   3.00000000e+00,
         1.56250000e+04])
```

[25]:
```
grader.check("q1_1")
```

[25]:
```
q1_1 results: All test cases passed!
```

Question 2. Make an array called `book_title_words` containing the following three strings: "Eats", "Shoots", and "and Leaves".

[29]:
```
book_title_words = make_array('Eats', 'Shoots', 'and Leaves')
book_title_words
```

[29]:
```
array(['Eats', 'Shoots', 'and Leaves'],
      dtype='<U10')
```

[30]:
```
grader.check("q1_2")
```

[30]:
```
q1_2 results: All test cases passed!
```

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 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, `"CS118".join(book_title_words)` .

[37]:
```
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
```

[38]:
```
grader.check("q1_3")
```

[38]:
```
q1_3 results: All test cases passed!
```

1.2 2. Indexing Arrays

These exercises give you practice accessing individual elements of arrays. In Python (and in many programming languages), each element is accessed by its *index*; for example, the first element is the element at index 0. Indices must be **integers**.

***Note:* If you have previous coding experience, you may be familiar with bracket notation. DO NOT use bracket notation when indexing (i.e. `arr[0]`), as this can yield different data type outputs than what we will be expecting. This can cause you to fail an autograder test.**

Be sure to refer to the Python Reference on the website if you feel stuck!

Question 1. The cell below creates an array of some numbers. Set `third_element` to the third element of `some_numbers`.

[41]:
```
some_numbers = make_array(-1, -3, -6, -10, -15)

third_element = some_numbers.item(2)
third_element
```

[41]:
```
-6
```

[42]:
```
grader.check("q2_1")
```

[42]:
```
q2_1 results: All test cases passed!
```

Question 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 (the cells that say "Ellipsis") by assigning `blank_a`, `blank_b`, `blank_c`, and `blank_d` to the correct elements in the table.

Hint: Replace the `...` with strings or numbers. As a reminder, indices should be **integers**.

[48]:
```
blank_a = "third"
blank_b = "fourth"
blank_c = 0
blank_d = 3
elements_of_some_numbers = Table().with_columns(
    "English name for position", make_array("first", "second", blank_a,␣
 ↪blank_b, "fifth"),
    "Index",                     make_array(blank_c, 1, 2, blank_d, 4),
    "Element",                   some_numbers)
elements_of_some_numbers
```

[48]:
```
English name for position | Index | Element
first                     | 0     | -1
second                    | 1     | -3
third                     | 2     | -6
fourth                    | 3     | -10
fifth                     | 4     | -15
```

[49]:
```
grader.check("q2_2")
```

[49]:
```
q2_2 results: All test cases passed!
```

Question 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?

[50]:
```
index_of_last_element = 141
```

[51]:
```
grader.check("q2_3")
```

[51]:
```
q2_3 results: All test cases passed!
```

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 an integer that represents the `len`gth of that array.

Question 4. The cell below loads an array called `president_birth_years`. Calling `tbl.column(...)` on a table returns an array of the column specified, in this case the `Birth Year` column of the `president_births` table. The last element in that array is the most recent among the birth years of all the deceased Presidents. Assign that year to `most_recent_birth_year`.

[65]:
```
president_birth_years = Table.read_table("president_births.csv").column('Birth␣
↪Year')

most_recent_birth_year = president_birth_years.item(37)
most_recent_birth_year
```

[65]:
```
1917
```

[66]:
```
grader.check("q2_4")
```

[66]:
```
q2_4 results: All test cases passed!
```

Question 5. Finally, assign `sum_of_birth_years` to the minimum of the first, sixteenth, and last birth years listed in `president_birth_years`.

[67]:
```
first_element = president_birth_years.item(0)
sixteenth_element = president_birth_years.item(15)
last_element = president_birth_years.item(37)

min_of_birth_years = min (first_element, sixteenth_element, last_element)
sum_of_birth_years = min_of_birth_years
sum_of_birth_years
```

[67]:
```
1732
```

[68]:
```
grader.check("q2_5")
```

[68]:
```
q2_5 results: All test cases passed!
```

1.3 3. Basic Array Arithmetic

Question 1. Multiply the numbers -42, 4224, 424224242, and -250 by 157. Assign each variable below such that `first_product` is assigned to the result of $-42 * 157$, `second_product` is assigned to the result of $4224 * 157$, and so on.

Note: For this question, **don't** use arrays.

[112]:
```
first_product = -42 * 157
second_product = 4224 * 157
third_product = 424224242 * 157
fourth_product = -250 * 157
print("First Product:", first_product)
print("Second Product:", second_product)
print("Third Product:", third_product)
print("Fourth Product:", fourth_product)
```

```
First Product: -6594
Second Product: 663168
```

```
Third Product: 66603205994
Fourth Product: -39250
```

[113]:
```
grader.check("q3_1")
```

[113]:
```
q3_1 results: All test cases passed!
```

Question 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`.

[116]:
```
numbers = make_array(-42, 4224, 424224242, -250)
products = numbers * 157
products
```

[116]:
```
array([      -6594,      663168, 66603205994,      -39250])
```

[117]:
```
grader.check("q3_2")
```

[117]:
```
q3_2 results: All test cases passed!
```

Question 3. Oops, we made a typo! Instead of 157, we wanted to multiply each number by 1577. Compute the correct 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.

[118]:
```
numbers = make_array(-42, 4224, 424224242, -250)
correct_products = numbers * 1577
correct_products
```

[118]:
```
array([      -66234,      6661248, 669001629634,      -394250])
```

[119]:
```
grader.check("q3_3")
```

[119]:
```
q3_3 results: All test cases passed!
```

Question 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 all the temperatures to Celsius by first subtracting 32 from them, then multiplying the results by $\frac{5}{9}$, i.e. $C = (F - 32) * \frac{5}{9}$. After converting the temperatures to Celsius, make sure to **ROUND** the final result to the nearest integer using the `np.round` function.

[120]:
```
max_temperatures = Table.read_table("temperatures.csv").column("Daily Max␣
↪Temperature")

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

[120]:
```
array([ -4.,  31.,  32., …,  17.,  23.,  16.])
```

[121]:
```
grader.check("q3_4")
```

[121]:
```
q3_4 results: All test cases passed!
```

Question 5. The cell below loads all the *lowest* temperatures from each day (in Fahrenheit). Compute the daily temperature range for each day. That is, compute the difference between each daily maximum temperature and the corresponding daily minimum temperature. **Pay attention to the units and give your answer in Celsius!** Make sure **NOT** to round your answer for this question!

Note: Remember that in Question 4, `celsius_max_temperatures` was rounded, so you might not want to use that variable in this question.

[126]:
```
min_temperatures = Table.read_table("temperatures.csv").column("Daily Min␣
 ↪Temperature")
celsius_min_temperatures = (min_temperatures - 32) * (5 / 9)
celsius_temperature_ranges = celsius_max_temperatures - celsius_min_temperatures
celsius_temperature_ranges
```

[126]:
```
array([  6.66666667,  10.        ,  12.22222222, …,  17.22222222,
        11.66666667,  11.11111111])
```

[127]:
```
grader.check("q3_5")
```

[127]:
```
q3_5 results: All test cases passed!
```

1.4 4. Old Faithful

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 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.

[128]:
```
waiting_times = Table.read_table('old_faithful.csv').column('waiting')

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

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 70.8970588235 minutes on
average.
```

[129]:
```
grader.check("q4_1")
```

[129]:
```
q4_1 results: All test cases passed!
```

Question 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: We want to return the absolute value of the biggest decrease.

Note: `np.diff()` calculates the difference between subsequent values in an array. Decreases are going to be negative, increases are going to be positive. For example, calling `np.diff()` on the array `make_array(1, 8, 3, 5)` evaluates to `array([8 - 1, 3 - 8, 5 - 3])`, or `array([7, -5, 2])`.

[131]:
```
# np.diff() calculates the difference between subsequent values
# in a NumPy array.
differences = np.diff(waiting_times)
biggest_decrease = np.min(differences)
biggest_decrease =  abs(biggest_decrease)
biggest_decrease
```

[131]:
```
45
```

[132]:
```
grader.check("q4_2")
```

[132]:
```
q4_2 results: All test cases passed!
```

Question 3. The `faithful_with_eruption_nums` table contains two columns: `eruption_number`, which represents the number of that eruption, and `waiting`, which represents the time spent waiting after that eruption. For example, take the first two rows of the table:

eruption number	waiting
1	79
2	54

We can read this as follows: after the first eruption, we waited 79 minutes for the second eruption. Then, after the second eruption, we waited 54 minutes for the third eruption.

Suppose Oscar and Wendy started watching Old Faithful at the start of the first eruption. Assume that they watch until the end of the tenth eruption. For some of that time they will be watching eruptions, and for the rest of the time they will be waiting for Old Faithful to erupt. How many minutes will they spend waiting for eruptions?

Hint #1: One way to approach this problem is to use the `take` or `where` method on the table `faithful_with_eruption_nums`.

Hint #2: `first_nine_waiting_times` must be an array.

[133]:
```
# The following two lines load in our faithful_with_eruption_nums table
faithful = Table.read_table('old_faithful.csv').drop("eruptions")
```

```
faithful_with_eruption_nums = faithful.with_column("eruption number", np.
 ↪arange(faithful.num_rows) + 1).select(1, 0)

first_nine_waiting_times = faithful_with_eruption_nums.take(np.arange(9)).
 ↪column('waiting')
total_waiting_time_until_tenth = sum(first_nine_waiting_times)
total_waiting_time_until_tenth
```

[133]:
```
633
```

[134]:
```
grader.check("q4_3")
```

[134]:
```
q4_3 results: All test cases passed!
```

Question 4. Let's imagine your guess for the next waiting time was always just the length of the previous waiting time. If you always guessed the previous waiting time, how big would your error in guessing the waiting times be, on average?

For example, since the first four waiting times are 79, 54, 74, and 62, the average difference between your guess and the actual time for just the second, third, and fourth eruptions would be $\frac{|79-54|+|54-74|+|74-62|}{3} = 19$.

[141]:
```
differences = np.diff(waiting_times)
absolute_differences = np.abs(np.diff(waiting_times))
average_error = np.mean(absolute_differences)
average_error
```

[141]:
```
20.520295202952031
```

[142]:
```
grader.check("q4_4")
```

[142]:
```
q4_4 results: All test cases passed!
```

1.5 5. Tables

Question 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`. Assign the new table to the variable `fruits`.

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

[146]:
```
# Our solution uses 1 statement split over 3 lines.
fruits = Table().with_columns(
    "fruit name", make_array("apple", "orange", "pineapple"),
    "count", make_array(4, 3, 3))

fruits
```

[146]:
```
fruit name | count
apple      | 4
orange     | 3
pineapple  | 3
```

[147]:
```
grader.check("q5_1")
```

[147]:
```
q5_1 results: All test cases passed!
```

Question 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` using the `Table.read_table()` function. `Table.read_table(...)` takes one argument (data file name in string format) and returns a table.

[148]:
```
inventory = Table.read_table("inventory.csv")
inventory
```

[148]:
```
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
```

[149]:
```
grader.check("q5_2")
```

[149]:
```
q5_2 results: All test cases passed!
```

Question 3. Does each box at the fruit stand contain a different fruit? Set `all_different` to `True` if each box contains a different fruit or to `False` if multiple boxes contain the same fruit.

Hint: You don't have to write code to calculate the True/False value for `all_different`. Just look at the `inventory` table and assign `all_different` to either `True` or `False` according to what you can see from the table in answering the question.

[150]:
```
all_different = False
all_different
```

[150]:
```
False
```

[151]:
```
grader.check("q5_3")
```

[151]:
```
q5_3 results: All test cases passed!
```

Question 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`.

[152]:
```
sales = Table.read_table("sales.csv")
sales
```

[152]:
```
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
```

[153]:
```
grader.check("q5_4")
```

[153]:
```
q5_4 results: All test cases passed!
```

Question 5. How many fruits did the store sell in total on that day?

[156]:
```
total_fruits_sold = sum(sales.column("count sold"))

total_fruits_sold
```

[156]:
```
638
```

[157]:
```
grader.check("q5_5")
```

[157]:
```
q5_5 results: All test cases passed!
```

Question 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.

[158]:
```
total_revenue = total_revenue = sum(sales.column("count sold") * sales.
↪column("price per fruit ($)"))
total_revenue
```

[158]:
```
106.84999999999999
```

[159]:
```
grader.check("q5_6")
```

[159]:
```
q5_6 results: All test cases passed!
```

Question 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 **original** count, so that the `count` column is **updated to be** the amount of fruit remaining after Saturday.

[160]:
```
remaining_counts = inventory.column("count") - sales.column("count sold")
remaining_inventory =  inventory.with_column("count", remaining_counts)
remaining_inventory
```

[160]:
```
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
```

[161]:
```
grader.check("q5_7")
```

[161]:
```
q5_7 results: All test cases passed!
```

1.6 Congratulations! You're done with Homework 2!

Be sure to run all of the cells and the grader checks and verify that they all pass, then choose **Download as PDF via LaTeX** from the **File** menu, as well as **Download as Notebook** from the **File** menu, correctly name your files, and submit the two files on **canvas**.