Homework 2: Arrays and Tables

```
# Don't change this cell; just run it.
import numpy as np
from datascience import *
```

Recommended Reading:

- Data Types
- Sequences
- Tables

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.

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.

Deadline:

This assignment is due **Monday, February 5th at 11:59pm**. No late assignments will be accepted.

Once your assignment is complete, do two things:

1. Change the file name by by adding "Submitted" at the end. For example, if I were a student submitting lab01, the file would initially be called "lab01Liz." After completing it, I would change the name to "lab01LizSubmitted."
2. Print your notebook as a pdf. Go to File -> Print -> save as pdf. Then upload the pdf to the corresponding assignment (lab/hw) on Canvas.

Directly sharing answers is not okay, but discussing problems with the instructor or with other students is encouraged. Refer to the policies page to learn more about how to learn cooperatively.

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

1. Creating Arrays

Question 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.

Note: Python lists are different/behave differently than numpy arrays. In this course, we use numpy arrays, so please make an **array**, not a python list.

```
# Our solution involved one extra line of code before creating
# weird_numbers.

import math
weird_numbers = make_array(-2, np.sin(1.2), 3, np.power(5, np.cos(1.2)))
#weird_numbers = np.array(-2, float(round(math.sin(1.2), 4)))
weird_numbers
```

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

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

```
book_title_words = np.array(["Eats", "Shoots", "and Leaves"])
book_title_words
```

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

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

```
blankString = " "
commaString = ","
with_commas = commaString.join(book_title_words)
without_commas = blankString.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
```

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.

Note: Please don't 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.

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

```
some_numbers = make_array(-1, -3, -6, -10, -15)

third_element = -10
third_element
```

```
-10
```

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.

```
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
```

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

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?

```
index_of_last_element = 141
```

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 4. The cell below loads an array called `president_birth_years`. Calling `.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 birth year of any deceased president. Assign that year to `most_recent_birth_year`.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
president_birth_years = Table.read_table('/content/drive/MyDrive/Colab/hw02/president_births
print(president_birth_years)

most_recent_birth_year = ...
most_recent_birth_year
```

```
Name                 | Birth      | Death      | Birth Year | Birth Days (since 1 CE)
George Washington    | 1732-02-22 | 1799-12-14 | 1732       | 632286
John Adams           | 1735-10-30 | 1826-07-04 | 1735       | 633632
Thomas Jefferson     | 1743-04-13 | 1826-07-04 | 1743       | 636354
James Madison        | 1751-03-16 | 1836-06-28 | 1751       | 639248
James Monroe         | 1758-04-28 | 1831-07-04 | 1758       | 641848
Andrew Jackson       | 1767-03-15 | 1845-06-08 | 1767       | 645091
John Quincy Adams    | 1767-07-11 | 1848-02-23 | 1767       | 645209
William Henry Harrison | 1773-02-09 | 1841-04-04 | 1773       | 647249
Martin Van Buren     | 1782-12-05 | 1862-07-24 | 1782       | 650835
Zachary Taylor       | 1784-11-24 | 1850-07-09 | 1784       | 651555
... (28 rows omitted)
Ellipsis
```

Question 5. Finally, assign `sum_of_birth_years` to the sum of the first, tenth, and last birth year in `president_birth_years`.

```
sum_of_birth_years = president_birth_years.column(3)[0] + president_birth_years.column(3)[9]
sum_of_birth_years
```

```
5433
```

3. Basic Array Arithmetic

Question 1. Multiply the numbers 42, 4224, 42422424, 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.

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

```
first_product = 42 * 157
second_product = 4224 * 157
third_product = 42422424 * 157
fourth_product = -250 * 157
print(first_product, second_product, third_product, fourth_product)
```

```
6594 663168 6660320568 -39250
```

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`.

```
numbers = make_array(42, 4224, 42422424, -250)
products = numbers * 157
products
```

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

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.

```
correct_products = numbers * 1577
correct_products
```

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

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 them all to Celsius by first subtracting 32 from them, then multiplying the results by $\frac{5}{9}$. Make sure to **ROUND** the final result after converting to Celsius to the nearest integer using the `np.round` function.

```
max_temperatures = Table.read_table("/content/drive/MyDrive/temperatures.csv").column("Daily
celsius_max_temperatures = (max_temperatures - 32) * 5/9
celsius_max_temperatures
```

```
array([ -3.88888889,  30.55555556,  31.66666667, ...,  16.66666667,
        22.77777778,  16.11111111])
```

Question 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. **Pay attention to the units, give your answer in Celsius!** Make sure **NOT** to round your answer for this question!

```
min_temperatures = Table.read_table("/content/drive/MyDrive/temperatures.csv").column("Daily

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

```
array([-11.11111111,  -7.77777778,  -5.55555556, ...,  -0.55555556,
        -6.11111111,  -6.66666667])
```

4. World Population

Remember that the tests from this point on will **not** necessarily tell you whether or not your answers are correct.

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.

```
world = Table.read_table("/content/drive/MyDrive/world_population.csv").select('Year', 'Popu
world.show(4)
```

Year	Population
1950	2557628654
1951	2594939877
1952	2636772306
1953	2682053389

... (62 rows omitted)

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

```
population = world.column(1)
population
```

```
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. Numpy is a module that is often used in Data Science!

The difference function `np.diff` subtracts each element in an array from the element after it within the array. 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 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.

```
largest_population_change = np.max(np.diff(population))
largest_population_change
```

```
87515824
```

Question 2. What do the values in the resulting array represent (choose one)?

```
np.cumsum(np.diff(population))
```

```
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.

```
# Assign cumulative_sum_answer to 1, 2, or 3
cumulative_sum_answer = 3
```

5. 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.

```
waiting_times = Table.read_table('/content/drive/MyDrive/old_faithful.csv').column('waiting'
shortest = min(waiting_times)
longest = max(waiting_times)
average = np.round(np.sum(waiting_times)/waiting_times.size)

print("Old Faithful erupts every", shortest, "to", longest, "minutes and every", average, "m
```

```
Old Faithful erupts every 43 to 96 minutes and every 71.0 minutes on average.
```

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 1: You'll need an array arithmetic function mentioned in the textbook. You have also seen this function earlier in the homework!

Hint 2: We want to return the absolute value of the biggest decrease.

```
biggest_decrease = max(np.abs(np.diff(waiting_times)))
biggest_decrease
```

```
47
```

Question 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. You'll go through `np.arange` more thoroughly in Lab 3, but you can read about it in this textbook section.

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 $|207 - 180| = 27$.

```
#difference_from_expected_easy = np.abs(sum(waiting_times)- 60 * waiting_times.size)
difference_from_expected = np.cumsum(waiting_times)- np.arange(0, 272*60, 60)
difference_from_expected
```

```
array([  79,   73,   87,   89,  114,  109,  137,  162,  153,  178,  172,
        196,  214,  201,  224,  216,  218,  242,  234,  253,  244,  231,
        249,  258,  272,  295,  290,  306,  324,  343,  356,  373,  379,
        399,  413,  405,  393,  413,  412,  442,  462,  460,  484,  482,
        495,  518,  522,  515,  537,  536,  551,  581,  575,  595,  589,
        612,  623,  627,  644,  665,  664,  688,  676,  698,  698,  730,
        748,  766,  771,  784,  806,  802,  821,  832,  834,  850,  850,
        868,  884,  907,  922,  944,  954,  959,  972, 1000, 1016, 1036,
       1024, 1050, 1050, 1080, 1070, 1088, 1091, 1103, 1127, 1142, 1133,
       1155, 1157, 1185, 1174, 1197, 1218, 1205, 1229, 1221, 1247, 1268,
       1283, 1282, 1311, 1330, 1329, 1350, 1340, 1365, 1364, 1391, 1384,
       1393, 1410, 1406, 1434, 1455, 1440, 1462, 1457, 1487, 1472, 1495,
       1491, 1520, 1506, 1528, 1519, 1545, 1538, 1557, 1578, 1578, 1600,
       1617, 1633, 1632, 1652, 1641, 1677, 1670, 1687, 1704, 1709, 1730,
       1741, 1751, 1772, 1805, 1798, 1827, 1812, 1838, 1836, 1854, 1860,
       1876, 1879, 1907, 1899, 1932, 1921, 1918, 1935, 1943, 1964, 1985,
       1998, 1988, 2013, 2027, 2022, 2039, 2062, 2085, 2076, 2094, 2118,
       2104, 2127, 2122, 2143, 2140, 2156, 2180, 2197, 2218, 2245, 2262,
       2253, 2271, 2271, 2293, 2324, 2317, 2335, 2321, 2338, 2362, 2351,
       2374, 2385, 2405, 2394, 2409, 2413, 2429, 2422, 2456, 2451, 2467,
       2457, 2479, 2473, 2488, 2506, 2525, 2543, 2561, 2571, 2590, 2600,
       2594, 2620, 2610, 2640, 2634, 2628, 2645, 2664, 2668, 2683, 2670,
       2696, 2699, 2724, 2746, 2743, 2765, 2772, 2786, 2780, 2803, 2816,
       2829, 2857, 2877, 2888, 2911, 2907, 2926, 2944, 2968, 2966, 2989,
       2972, 2972, 2987, 3008, 2994, 3024, 3010, 3024])
```

Question 4. Let's imagine your guess for the next wait 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 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 $\frac{|79-54|+|54-74|}{2} = 22.5$.

```
total_difference = 0;
for i in range (waiting_times.size-1):
    total_difference += waiting_times[i] - waiting_times[i+1]

average_error = total_difference / 2
average_error
```

```
2.5
```

6. 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"`.

```
# Our solution uses 1 statement split over 3 lines.
fruits = [["fruit name", "count"],
          ["apples", 4],
          ["oranges", 3],
          ["pineapples", 3]]
fruits
```

```
[['fruit name', 'count'], ['apples', 4], ['oranges', 3], ['pineapples', 3]]
```

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.

```
inventory = Table.read_table('/content/drive/MyDrive/inventory.csv')
inventory
```

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

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.

```
all_different = False
all_different
```

```
False
```

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`.

```
sales = Table.read_table('/content/drive/MyDrive/sales.csv')
sales
```

box ID	fruit name	count sold	price per fruit ($)
53686	kiwi	3	0.5

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

```
total_fruits_sold = sum(sales.column(2))
total_fruits_sold
```

```
638
```

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.

```
total_revenue = sum(sales.column(2)*sales.column(3))
total_revenue
```

```
106.84999999999999
```

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 count, so that the "count" is the amount of fruit remaining after Saturday.

```
remaining_inventory = inventory
remaining_inventory["count"] = inventory.column(2) - sales.column(2)
remaining_inventory
```

box ID	fruit name	count
53686	kiwi	42
57181	strawberry	22
25274	apple	20
48800	orange	0
26187	strawberry	230
57930	grape	162