Homework 2: Arrays and Tables

Please complete this notebook by filling in the cells provided. It is recommended (but not required) to do Practice 02 (https://github.com/YData123/sds123-sp21/raw/main/practice_exercises/practice02.zip) before this homework, since some functions/methods mentioned in Practice 02 might be useful for this homework.

Recommended Reading:

- Data Types (https://www.inferentialthinking.com/chapters/04/data-types.html)
- Sequences (https://www.inferentialthinking.com/chapters/05/sequences.html)
- Tables (https://www.inferentialthinking.com/chapters/06/tables.html)

Deadline:

Homework 2 is due Thursday, February 18 at 11:59pm. Late work will not be accepted as per the course policies (see the Syllabus and Course policies on Canvas (https://canvas.yale.edu).

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. The drop-in office hours schedule can be found on Canvas (https://canvas.yale.edu). You can also post questions or start discussions on Ed Discussion (https://edstem.org/us/courses/3558/discussion/).

Submission:

Submit your assignment as a .pdf on Gradescope. You can access Gradescope through Canvas on the left-side of the class home page. The problems in each homework assignment are numbered. NOTE: When submitting on Gradescope, please select the correct pages of your pdf that correspond to each problem. This will allow graders to find your complete solution to each problem.

To produce the .pdf, please do the following in order to preserve the cell structure of the notebook:

1. Go to "File" at the top-left of your Jupyter Notebook
2. Under "Download as", select "HTML (.html)"
3. After the .html has downloaded, open it and then select "File" and "Print" (note you will not actually be printing)
4. From the print window, select the option to save as a .pdf

In [1]:
```
# Don't change this cell; just run it.

import numpy as np
from datascience import *
```

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.

In [ ]:
```
# Our solution involved one extra line of code before creating
# weird_numbers.
...
weird_numbers = ...
weird_numbers
```

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

In [ ]:
```
book_title_words = ...
book_title_words
```

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 (https://en.wikipedia.org/wiki/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, `"yale".join(book_title_words)`.

In [3]:
```
with_commas = ...
without_commas = ...

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

```
with_commas: Ellipsis
without_commas: Ellipsis
```

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*. The first element has an index of 0. The second element has an index of 1.

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

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

third_element = ...
third_element
```

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.

In [ ]:
```
elements_of_some_numbers = Table().with_columns(
    "English name for position", make_array("first", "second", "???",
"???", "fifth"),
    "Index",                     make_array("???", "1", "2", "???", "4"
),
    "Element",                   some_numbers)
elements_of_some_numbers
```

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?

In [ ]:
```
index_of_last_element = ...
```

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 as of 2017. Assign that year to `most_recent_birth_year`. (*Hint*: Use the function `len`.)

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

most_recent_birth_year = ...
most_recent_birth_year
```

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

In [ ]:
```
sum_of_birth_years = ...
sum_of_birth_years
```

3. Basic Array Arithmetic

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

In [ ]:
```
first_product = ...
second_product = ...
third_product = ...
fourth_product = ...
print(first_product, second_product, third_product, fourth_product)
```

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` .

In [ ]:
```
numbers = ...
products = ...
products
```

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 `numbers` containing the 4 numbers.

In [ ]:
```
fixed_products = ...
fixed_products
```

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 (https://www.noaa.gov), 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.

In [ ]:
```
max_temperatures = Table.read_table("temperatures.csv").column("Daily Ma
x Temperature")

celsius_max_temperatures = ...
celsius_max_temperatures
```

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!

In [ ]:
```
min_temperatures = Table.read_table("temperatures.csv").column("Daily Mi
n Temperature")

celsius_temperature_ranges = ...
celsius_temperature_ranges
```

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 (https://www.census.gov/en.html).

In [4]:
```
world = Table.read_table("world_population.csv").select('Year', 'Populat
ion')
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.

In [5]:
```
population = world.column(1)
population
```

Out[5]:
```
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.

In [ ]:
```
largest_population_change = ...
largest_population_change
```

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

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

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 (with a total change of 0).

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

5. Old Faithful

Old Faithful is a geyser in Yellowstone that erupts every 44 to 125 minutes (according to Wikipedia (https://en.wikipedia.org/wiki/Old_Faithful)). People are often told that the geyser erupts every hour (http://yellowstone.net/geysers/old-faithful/), 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. (*Hint:* You can round average waiting time to 3 decimal places.)

In [6]:
```
waiting_times = Table.read_table('old_faithful.csv').column('waiting')

shortest = ...
longest = ...
average = ...

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

```
Old Faithful erupts every Ellipsis to Ellipsis minutes and every Ellips
is minutes on average.
```

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: The function you use may report positive or negative values. You will have to determine if the biggest decrease corresponds to the highest or lowest value. Ultimately, we want to return the absolute value of the biggest decrease so the final answer is positive.

In [ ]:
```
biggest_decrease = ...
biggest_decrease
```

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.

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 $|207 - 180| = 27$.

Hint: You'll need to compare cumulative sum to a range. `np.cumsum` and `np.arange` might be useful.

In [ ]:
```
difference_from_expected = ...
difference_from_expected
```

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 $\frac{|79-54|+|54-74|}{2} = 22.5$.

In [ ]:
```
average_error = ...
average_error
```

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"`.

In [ ]:
```
# Our solution uses 1 statement split over 3 lines.
fruits = ...
         ...
         ...
fruits
```

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`.

In [ ]:
```
inventory = ...
inventory
```

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

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

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`.

In [ ]:
```
sales = ...
sales
```

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

In [ ]:
```
total_fruits_sold = ...
total_fruits_sold
```

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 for grape.

In [ ]:
```
total_revenue = ...
total_revenue
```

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.

In [ ]:
```
remaining_inventory = ...
    ...
    ...
    ...
remaining_inventory
```

7. Submission

Once you're finished, submit your assignment as a .pdf (download as .html, then print to save as a .pdf) on Gradescope.