Homework 2: Arrays and Tables

In [1]:
```
# Don't change this cell; just run it.
# When you log-in please hit return (not shift + return) after typing in
your email
import numpy as np
from datascience import *
_ = ok.auth(inline=True)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call l
ast)
<ipython-input-1-ee98e362a7ed> in <module>
      3 import numpy as np
      4 from datascience import *
----> 5 _ = ok.auth(inline=True)

NameError: name 'ok' is not defined
```

Recommended Reading:

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

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.

This assignment is due Thursday, February 6 at 11:59 P.M. You will receive an early submission bonus point if you turn in your final submission by Wednesday, February 5 at 11:59 P.M. Late work will not be accepted as per the [policies (http://data8.org/sp20/policies.html)](http://data8.org/sp20/policies.html) of this course.

Start early so that you can come to office hours if you're stuck. Check the website for the [office hours schedule (http://data8.org/sp20/office-hours.html)](http://data8.org/sp20/office-hours.html).

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.

Important: In this homework, the `ok` tests will tell you whether your answer is correct, except for Parts 4, 5 & 6. In future homework assignments, correctness tests will typically not be provided.

Before continuing the assignment, select "Save and Checkpoint" in the File menu and then execute the submit cell below. The result will contain a link that you can use to check that your assignment has been submitted successfully. If you submit more than once before the deadline, we will only grade your final submission. If you mistakenly submit the wrong one, you can head to okpy.org and flag the correct version. There will be another submit cell at the end of the assignment when you finish!

In [2]:
```
# When you log-in please hit return (not shift + return) after typing in
your email
_ = ok.submit()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call l
ast)
<ipython-input-2-8db039f4ac7c> in <module>
      1 # When you log-in please hit return (not shift + return) after
 typing in your email
----> 2 _ = ok.submit()

NameError: name 'ok' is not defined
```

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.

```
BEGIN QUESTION
name: q1_1
```

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

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

```
In [4]: # TEST
        np.allclose(weird_numbers, np.array([-2.,  0.93203909,  3.,  1.79174913
        ]), rtol=1e-03, atol=1e-03)
```

```
Out[4]: True
```

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

```
BEGIN QUESTION
name: q1_2
```

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

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

```
In [6]: # TEST
        import numpy as np
        # It looks like you didn't make an array.
        type(book_title_words) == np.ndarray
```

```
Out[6]: True
```

```
In [7]: # TEST
        # It looks like you included commas in the text.
        # The three pieces of text in the array should be:
        #   "Eats"
        #   "Shoots"
        #   "and Leaves"
        not any([',' in text for text in book_title_words])
```

```
Out[7]: True
```

```
In [8]: # TEST
        # It looks like you didn't include both words in the
        # last piece of text.  It should be the actual string:
        #   "and Leaves"
        'and ' in book_title_words.item(2)
```

```
Out[8]: True
```

```
In [9]: # TEST
        len(book_title_words)
```

```
Out[9]: 3
```

```
In [10]: # TEST
         book_title_words.item(0) == 'Eats'
```

```
Out[10]: True
```

In [11]:
```
# TEST
book_title_words.item(1) == 'Shoots'
```
Out[11]:
```
True
```

In [12]:
```
# TEST
book_title_words.item(2) == 'and Leaves'
```
Out[12]:
```
True
```

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)](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 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)`.

```
BEGIN QUESTION
name: q1_3
```

In [13]:
```
with_commas = ", ".join(book_title_words) #SOLUTION
without_commas = " ".join(book_title_words) #SOLUTION

# 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
```

In [14]:
```
# TEST
with_commas
```
Out[14]:
```
'Eats, Shoots, and Leaves'
```

In [15]:
```
# TEST
without_commas
```
Out[15]:
```
'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.

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

```
BEGIN QUESTION
name: q2_1
```

In [16]:
```
some_numbers = make_array(-1, -3, -6, -10, -15)

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

Out[16]: -6

In [17]:
```
# TEST
# It looks like you wrote:
#   some_numbers.item(3)
# But the third element has index 2, not 3.
third_element != -10
```

Out[17]: True

In [18]:
```
# TEST
third_element
```

Out[18]: -6

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.

```
BEGIN QUESTION
name: q2_2
```

In [19]:
```
blank_a = "third" #SOLUTION
blank_b = "fourth" #SOLUTION
blank_c = 0 #SOLUTION
blank_d = 3 #SOLUTION
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
```

Out[19]:

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

In [20]:
```
# TEST
elements_of_some_numbers.column(0).item(2) == "third"
```

Out[20]: True

In [21]:
```
# TEST
elements_of_some_numbers.column(0).item(3) == "fourth"
```

Out[21]: True

In [22]:
```
# TEST
elements_of_some_numbers.column(1).item(0) == 0
```

Out[22]: True

In [23]:
```
# TEST
elements_of_some_numbers.column(1).item(3) == 3
```

Out[23]: True

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?

```
BEGIN QUESTION
name: q2_3
```

In [24]:
```
index_of_last_element = 142 - 1 # (or just 141) #SOLUTION
```

In [25]:
```
# TEST
index_of_last_element
```

Out[25]: 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`.

```
BEGIN QUESTION
name: q2_4
```

In [26]:
```
president_birth_years = Table.read_table("president_births.csv").column(
'Birth Year')

most_recent_birth_year = president_birth_years.item(len(president_birth_
years)-1) #SOLUTION
most_recent_birth_year
```

Out[26]: 1917

In [27]:
```
# TEST
most_recent_birth_year
```

Out[27]: 1917

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

```
BEGIN QUESTION
name: q2_5
```

In [28]:
```
sum_of_birth_years = sum(make_array(president_birth_years.item(0), presi
dent_birth_years.item(9), president_birth_years.item(len(president_birth
_years) - 1))) #SOLUTION
```

In [29]:
```
# TEST
sum_of_birth_years
```

Out[29]: 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.

```
BEGIN QUESTION
name: q3_1
```

In [30]:
```
first_product = 42 * 157 #SOLUTION
second_product = 4224 * 157 #SOLUTION
third_product = 42422424 * 157 #SOLUTION
fourth_product = -250 * 157 #SOLUTION
print(first_product, second_product, third_product, fourth_product)
```

```
6594 663168 6660320568 -39250
```

In [31]:
```
# TEST
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` .

```
BEGIN QUESTION
name: q3_2
```

In [32]:
```
numbers = make_array(42, 4224, 42422424, -250) #SOLUTION
products = numbers * 157 #SOLUTION
products
```

Out[32]:
```
array([      6594,     663168, 6660320568,     -39250])
```

In [33]:
```
# TEST
np.allclose(products, np.array([6594, 663168, 6660320568, -39250]))
```

Out[33]:
```
True
```

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.

```
BEGIN QUESTION
name: q3_3
```

In [34]:
```
correct_products = numbers * 1577 # SOLUTION
correct_products
```

Out[34]:
```
array([      66234,     6661248, 66900162648,     -394250])
```

In [35]:
```
# TEST
np.allclose(correct_products, np.array([66234, 6661248, 66900162648, -39
4250]))
```

Out[35]:
```
True
```

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 (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** the final result after converting to Celsius to the nearest integer using the `np.round` function.

```
BEGIN QUESTION
name: q3_4
```

In [36]:
```
max_temperatures = Table.read_table("temperatures.csv").column("Daily Ma
x Temperature")

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

Out[36]:
```
array([-4., 31., 32., ..., 17., 23., 16.])
```

In [37]:
```
# TEST
# It looks like you multiplied and subtracted in the wrong
# order.
sum(celsius_max_temperatures) != 356705.0
```

Out[37]:
```
True
```

In [38]:
```
# TEST
sum(celsius_max_temperatures)
```

Out[38]:
```
1280677.0
```

In [39]:
```
# TEST
len(celsius_max_temperatures)
```

Out[39]:
```
65000
```

In [40]:
```
# TEST
celsius_max_temperatures.item(2003)
```

Out[40]:
```
20.0
```

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!

```
BEGIN QUESTION
name: q3_5
```

In [41]:
```
min_temperatures = Table.read_table("temperatures.csv").column("Daily Mi
n Temperature")

celsius_temperature_ranges = (5/9) * (max_temperatures - min_temperature
s) #SOLUTION
celsius_temperature_ranges
```

Out[41]:
```
array([ 6.66666667, 10.        , 12.22222222, ..., 17.22222222,
       11.66666667, 11.11111111])
```

In [42]:
```
# TEST
round(sum(celsius_temperature_ranges))
```

Out[42]:
```
768487.0
```

In [43]:
```
# TEST
len(celsius_temperature_ranges)
```

Out[43]:
```
65000
```

In [44]:
```
# TEST
celsius_temperature_ranges.item(1)
```

Out[44]:
```
10.0
```

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

In [45]:
```
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 [46]:
```
population = world.column(1)
population
```

Out[46]:
```
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.

```
BEGIN QUESTION
name: q4_1
```

In [47]:
```
largest_population_change = max(np.diff(population)) #SOLUTION
largest_population_change
```

Out[47]: 87515824

In [48]:
```
# TEST
largest_population_change
```

Out[48]: 87515824

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

In [49]:
```
np.cumsum(np.diff(population))
```

Out[49]:
```
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.

```
BEGIN QUESTION
name: q4_2
```

In [50]:
```
# Assign cumulative_sum_answer to 1, 2, or 3
cumulative_sum_answer = 2 # SOLUTION
```

In [51]:
```
# TEST
type(cumulative_sum_answer) == int
```

Out[51]: True

In [52]:
```
# TEST
1 <= cumulative_sum_answer <= 3
```

Out[52]: True

In [53]:
```
# HIDDEN TEST
cumulative_sum_answer == 2
```

Out[53]: True

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)](https://en.wikipedia.org/wiki/Old_Faithful)). People are [often told that the geyser erupts every hour (http://yellowstone.net/geysers/old-faithful/)](http://yellowstone.net/geysers/old-faithful/), 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.

```
BEGIN QUESTION
name: q5_1
```

In [54]:
```
waiting_times = Table.read_table('old_faithful.csv').column('waiting')

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

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.
```

In [55]:
```
# TEST
# Hint: shortest is a number between 40 and 50.
40 <= shortest <= 50
```

Out[55]: True

In [56]:
```
# TEST
# Hint: the average is between the shortest and the longest
shortest <= average <= longest
```

Out[56]: True

In [57]:
```
# TEST
# Hint: the average is between the shortest and the longest
shortest <= average <= longest
```

Out[57]: True

In [58]:
```
# HIDDEN TEST
shortest
```

Out[58]: 43

In [59]:
```
# HIDDEN TEST
longest
```

Out[59]: 96

In [60]:
```
# HIDDEN TEST
np.isclose(average, 70.8970588235)
```

Out[60]: True

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 (https://www.inferentialthinking.com/chapters/05/1/arrays.html#Functions-on-Arrays)](https://www.inferentialthinking.com/chapters/05/1/arrays.html#Functions-on-Arrays). You have also seen this function earlier in the homework!

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

```
BEGIN QUESTION
name: q5_2
```

In [61]:
```
biggest_decrease = abs(min((np.diff(waiting_times)))) # SOLUTION
biggest_decrease
```

Out[61]: 45

In [62]:
```
# TEST
# Hint: If you are getting 47 as your answer, you might be computing the biggest change
# rather than the biggest decrease!
biggest_decrease == 47
```

Out[62]: False

In [63]:
```
# TEST
# Hint: biggest decrease is above 30, but not 47.
30 <= biggest_decrease < 47
```

Out[63]: True

In [64]:
```
# HIDDEN TEST
biggest_decrease
```

Out[64]: 45

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 (https://www.inferentialthinking.com/chapters/05/2/Ranges.html)](https://www.inferentialthinking.com/chapters/05/2/Ranges.html).

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 $|207 - 180| = 27$.

```
BEGIN QUESTION
name: q5_3
```

In [65]:
```
difference_from_expected = np.abs(np.cumsum(waiting_times) - np.arange(1
,273)*60) # SOLUTION
difference_from_expected
```

Out[65]:
```
array([  19,   13,   27,   29,   54,   49,   77,  102,   93,  118,  11
2,
        136,  154,  141,  164,  156,  158,  182,  174,  193,  184,  17
1,
        189,  198,  212,  235,  230,  246,  264,  283,  296,  313,  31
9,
        339,  353,  345,  333,  353,  352,  382,  402,  400,  424,  42
2,
        435,  458,  462,  455,  477,  476,  491,  521,  515,  535,  52
9,
        552,  563,  567,  584,  605,  604,  628,  616,  638,  638,  67
0,
        688,  706,  711,  724,  746,  742,  761,  772,  774,  790,  79
0,
        808,  824,  847,  862,  884,  894,  899,  912,  940,  956,  97
6,
        964,  990,  990, 1020, 1010, 1028, 1031, 1043, 1067, 1082, 107
3,
       1095, 1097, 1125, 1114, 1137, 1158, 1145, 1169, 1161, 1187, 120
8,
       1223, 1222, 1251, 1270, 1269, 1290, 1280, 1305, 1304, 1331, 132
4,
       1333, 1350, 1346, 1374, 1395, 1380, 1402, 1397, 1427, 1412, 143
5,
       1431, 1460, 1446, 1468, 1459, 1485, 1478, 1497, 1518, 1518, 154
0,
       1557, 1573, 1572, 1592, 1581, 1617, 1610, 1627, 1644, 1649, 167
0,
       1681, 1691, 1712, 1745, 1738, 1767, 1752, 1778, 1776, 1794, 180
0,
       1816, 1819, 1847, 1839, 1872, 1861, 1858, 1875, 1883, 1904, 192
5,
       1938, 1928, 1953, 1967, 1962, 1979, 2002, 2025, 2016, 2034, 205
8,
       2044, 2067, 2062, 2083, 2080, 2096, 2120, 2137, 2158, 2185, 220
2,
       2193, 2211, 2211, 2233, 2264, 2257, 2275, 2261, 2278, 2302, 229
1,
       2314, 2325, 2345, 2334, 2349, 2353, 2369, 2362, 2396, 2391, 240
7,
       2397, 2419, 2413, 2428, 2446, 2465, 2483, 2501, 2511, 2530, 254
0,
       2534, 2560, 2550, 2580, 2574, 2568, 2585, 2604, 2608, 2623, 261
0,
       2636, 2639, 2664, 2686, 2683, 2705, 2712, 2726, 2720, 2743, 275
6,
       2769, 2797, 2817, 2828, 2851, 2847, 2866, 2884, 2908, 2906, 292
9,
       2912, 2912, 2927, 2948, 2934, 2964, 2950, 2964])
```

In [66]:
```
# TEST
difference_from_expected.size
```

Out[66]: 272

In [67]:
```
# TEST
difference_from_expected.item(271) == abs(60 * 272 - sum(waiting_times))
```

Out[67]: True

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

```
BEGIN QUESTION
name: q5_4
```

In [68]:
```
average_error = np.average(abs(np.diff(waiting_times))) # SOLUTION
average_error
```

Out[68]: 20.52029520295203

In [69]:
```
# TEST
15 <= average_error <= 25
```

Out[69]: True

In [70]:
```
# HIDDEN TEST
np.isclose(average_error, 20.520295202952031)
```

Out[70]: True

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"`.

```
BEGIN QUESTION
name: q6_1
```

In [71]:
```
# Our solution uses 1 statement split over 3 lines.
fruits = Table().with_columns( #SOLUTION
        "fruit name", make_array("apple", "orange", "pineapple"), #SOLUT
ION
        "count",      make_array(4,       3,         3)) #SOLUTION
fruits
```

Out[71]:

fruit name	count
apple	4
orange	3
pineapple	3

In [72]:
```
# TEST
fruits.sort(0)
```

Out[72]:

fruit name	count
apple	4
orange	3
pineapple	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.

```
BEGIN QUESTION
name: q6_2
```

In [73]:
```
inventory = Table.read_table("inventory.csv") #SOLUTION
inventory
```

Out[73]:

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

In [74]:
```
# TEST
inventory.sort(0).column(0).item(0)
```
Out[74]:
```
25274
```

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.

```
BEGIN QUESTION
name: q6_3
```

In [75]:
```
all_different = False #SOLUTION
all_different
```
Out[75]:
```
False
```

In [76]:
```
# TEST
all_different in {True, False}
```
Out[76]:
```
True
```

In [77]:
```
# HIDDEN TEST
all_different == False
```
Out[77]:
```
True
```

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` .

```
BEGIN QUESTION
name: q6_4
```

In [78]:
```
sales = Table.read_table("sales.csv") #SOLUTION
sales
```
Out[78]:

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

In [79]:
```
# TEST
sales.sort(0)
```

Out[79]:

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

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

```
BEGIN QUESTION
name: q6_5
```

In [80]:
```
total_fruits_sold = sum(sales.column("count sold")) #SOLUTION
total_fruits_sold
```

Out[80]: 638

In [81]:
```
# TEST
# We're asking for the number of *pieces* of fruit, not the
# number of kinds of fruit or the number of boxes from which
# there were sales.
total_fruits_sold > 10
```

Out[81]: True

In [82]:
```
# HIDDEN TEST
total_fruits_sold
```

Out[82]: 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.

```
BEGIN QUESTION
name: q6_6
```

In [83]:
```
total_revenue = sum(sales.column("count sold") * sales.column("price per fruit ($)")) #SOLUTION
total_revenue
```

Out[83]: 106.85

In [84]:
```
# TEST
# If you're stuck, here's a hint: You want to multiply the count
# sold in each box by the per-item price of fruits in that box.
# You can use elementwise multiplication for that.
# Then you want the sum of those products.  Use sum().
50 <= total_revenue <= 150
```

Out[84]: True

In [85]:
```
# HIDDEN TEST
np.isclose(total_revenue, 106.85)
```

Out[85]: True

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.

```
BEGIN QUESTION
name: q6_7
```

In [86]:
```
remaining_inventory = Table().with_columns( #SOLUTION
    "box ID", inventory.column("box ID"), #SOLUTION
    "fruit name", inventory.column("fruit name"), #SOLUTION
    "count", inventory.column("count") - sales.column("count sold")) #SOLUTION
remaining_inventory
```

Out[86]:

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

In [87]:
```
# TEST
# It looks like your table doesn't have all 3 columns that are
# in the inventory table.
remaining_inventory.num_columns
```

Out[87]: 3

In [88]:
```
# TEST
#It looks like you forgot to subtract off the sales.
remaining_inventory.column("count").item(0) != 45
```

Out[88]: True

In [89]:
```
# TEST
remaining_inventory.where(1, 'grape')
```

Out[89]:

box ID	fruit name	count
57930	grape	162

7. Submission

Once you're finished, select "Save and Checkpoint" in the File menu and then execute the `submit` cell below. The result will contain a link that you can use to check that your assignment has been submitted successfully. If you submit more than once before the deadline, we will only grade your final submission. If you mistakenly submit the wrong one, you can head to okpy.org (https://okpy.org/) and flag the correct version. To do so, go to the website, click on this assignment, and find the version you would like to have graded. There should be an option to flag that submission for grading!

In [90]:
```
_ = ok.submit()
```

```
---------------------------------------------------------------------------
----
NameError                                 Traceback (most recent call l
ast)
<ipython-input-90-cc46ca874451> in <module>
----> 1 _ = ok.submit()

NameError: name 'ok' is not defined
```

In [91]:
```
# For your convenience, you can run this cell to run all the tests at on
ce!
import os
print("Running all tests...")
_ = [ok.grade(q[:-3]) for q in os.listdir("tests") if q.startswith('q')
and len(q) <= 10]
print("Finished running all tests.")
```

```
Running all tests...
Finished running all tests.
```