hw02-solution

February 19, 2020

1 Homework 2: Arrays and Tables

Reading: Textbook chapters 4 and 5.

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.

Homework 2 is due Tuesday, 25 Feb at 11:59pm. Start early so that you can come to office hours if you're stuck. Check the website for the office hours schedule. It is much better to use https://mzareei.youcanbook.me/.

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

import numpy as np
from datascience import *

from client.api.notebook import Notebook
ok = Notebook('hw02.ok')
_ = ok.auth(inline=True)
```

```
=====================================================================
Assignment: Homework 2: Arrays and Tables
OK, version v1.14.20
=====================================================================


Successfully logged in as m.zareei@ieee.org
```

Important: The `ok` tests don't always tell you that your answer is correct. More often, they help catch careless mistakes. It's up to you to ensure that your answer is correct. If you're not sure, ask someone (not for the answer, but for some guidance about your approach).

Once you're finished, select "Save and Checkpoint" in the File menu and then execute the `submit` cell below. Check your Okpy account to see if your backup has saved successfully. If you submit more than once before the deadline, we will only grade your final submission.

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

1.1 1. Studying the Survivors

The Reverend Henry Whitehead was skeptical of John Snow's conclusion about the Broad Street pump. After the Broad Street cholera epidemic ended, Whitehead set about trying to prove Snow wrong. (The history of the event is detailed here.)

He realized that Snow had focused his analysis almost entirely on those who had died. Whitehead, therefore, investigated the drinking habits of people in the Broad Street area who had not died in the outbreak.

What is the main reason it was important to study this group?

1) If Whitehead had found that many people had drunk water from the Broad Street pump and not caught cholera, that would have been evidence against Snow's hypothesis.

2) Survivors could provide additional information about what else could have caused the cholera, potentially unearthing another cause.

3) Through considering the survivors, Whitehead could have identified a cure for cholera.

[2]:
```
# Set survivor_answer to 1, 2, or 3
survivor_answer = 1 # SOLUTION
```

[3]:
```
_ = ok.grade('q1_1')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooook] 100.0% passed
```

Note: Whitehead ended up finding further proof that the Broad Street pump played the central role in spreading the disease to the people who lived near it. Eventually, he became one of Snow's greatest defenders.

1.2 2. 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.

[4]:
```
# 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)) #SOLUTION
weird_numbers
```

[4]:
```
array([-2.        ,  0.93203909,  3.        ,  1.79174913])
```

[5]:
```
_ = ok.grade('q2_1')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooook] 100.0% passed
```

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

[17]:
```
book_title_words = make_array("Eats", "Shoots", "and Leaves") #SOLUTION
book_title_words
```

[17]:
```
array(['Eats', 'Shoots', 'and Leaves'], dtype='<U10')
```

[18]:
```
_ = ok.grade('q2_2')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Question 2_2 > Suite 1 > Case 1

>>> import numpy as np
>>> # It looks like you didn't make an array.
>>> type(book_title_words) == np.ndarray
True
>>> # 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])
True
>>> # 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)
True
>>> book_title_words
array(['Eats', 'Shoots', 'and Leaves'], dtype='<U10')

# Error: expected
#     array(['Eats', 'Shoots', 'and Leaves'],
#           dtype='<U10')
# but got
#     array(['Eats', 'Shoots', 'and Leaves'], dtype='<U10')

Run only this test case with "python3 ok -q q2_2 --suite 1 --case 1"
---------------------------------------------------------------------
Test summary
    Passed: 0
    Failed: 1
[k…] 0.0% 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, `"foo".join(book_title_words)` .

[19]:
```
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
```

[20]:
```
_ = ok.grade('q2_3')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
```

```
Test summary
    Passed: 1
    Failed: 0
[oooooooooook] 100.0% passed
```

1.3 3. 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`.

[21]:
```
some_numbers = make_array(-1, -3, -6, -10, -15)

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

[21]:
```
-6
```

[22]:
```
_ = ok.grade('q3_1')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooook] 100.0% 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 in the cell (the strings that are currently "???") to complete the table.

[27]:
```
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
```

[27]:
```
English name for position | Index | Element
first                     | 0     | -1
second                    | 1     | -3
```

```
third                    | 2     | -6
fourth                   | 3     | -10
fifth                    | 4     | -15
```

[28]:
```
_ = ok.grade('q3_2')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooook] 100.0% 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?

[29]:
```
index_of_last_element = 142 - 1 # (or just 141) #SOLUTION
```

[30]:
```
_ = ok.grade('q3_3')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooook] 100.0% 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 the `len`gth of that array (an integer).

Question 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`.

[32]:
```
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
```

[32]: 1917

[33]:
```
_ = ok.grade('q3_4')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooook] 100.0% passed
```

1.4 4. Basic Array Arithmetic

Question 1. Multiply the numbers 42, 4224, 42422424, and -250 by 157. For this question, **don't** use arrays.

[34]:
```
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
```

[35]:
```
_ = ok.grade('q4_1')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooook] 100.0% 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`.

[36]:
```
numbers = make_array(42, 4224, 42422424, -250) #SOLUTION
products = numbers * 157 #SOLUTION
products
```

[36]:
```
array([      6594,     663168, 6660320568,     -39250])
```

[37]:
```
_ = ok.grade('q4_2')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooook] 100.0% passed
```

Question 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.

[38]:
```
fixed_products = numbers * 1577 # SOLUTION
fixed_products
```

[38]:
```
array([      66234,     6661248, 66900162648,     -394250])
```

[39]:
```
_ = ok.grade('q4_3')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooook] 100.0% 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 them all to Celsius by first subtracting 32 from them, then multiplying the results by $\frac{5}{9}$. Round each result to the nearest integer using the `np.round` function.

[40]:
```
max_temperatures = Table.read_table("temperatures.csv").column("Daily Max␣
↪Temperature")

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

[40]:
```
array([-4., 31., 32., …, 17., 23., 16.])
```

[41]:
```
_ = ok.grade('q4_4')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests
```

```
-------------------------------------------------------------------
Test summary
    Passed: 3
    Failed: 0
[oooooooooook] 100.0% passed
```

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. **Give your answer in Celsius!**

[42]:
```
min_temperatures = Table.read_table("temperatures.csv").column("Daily Min␣
↪Temperature")

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

[42]:
```
array([ 6.66666667, 10.        , 12.22222222, …, 17.22222222,
       11.66666667, 11.11111111])
```

[43]:
```
_ = ok.grade('q4_5')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

-------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed
```

1.5 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.

[44]:
```
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.
```

[45]:
```
_ = ok.grade('q5_1')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 3
    Failed: 0
[ooooooooook] 100.0% passed
```

Question 2. Assign `biggest_decrease` to the biggest descrease 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.

[46]:
```
biggest_decrease = -min(np.diff(waiting_times)) # SOLUTION
biggest_decrease
```

[46]:
```
45
```

[47]:
```
_ = ok.grade('q5_2')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed
```

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.

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 $|207 - 180| = 27$.

[68]:
```
difference_from_expected = np.abs(np.cumsum(waiting_times)) - (np.
↪arange(272)+1)*60 # SOLUTION
difference_from_expected
```

[68]:
```
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])
```

[69]:
```
_ = ok.grade('q5_3')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooook] 100.0% passed
```

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

[70]:
```
average_error = np.average(abs(np.diff(waiting_times))) # SOLUTION
average_error
```

[70]:
```
20.52029520295203
```

[71]:
```
_ = ok.grade('q5_4')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooo] 100.0% passed
```

1.6 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". Give it the name `fruits`.

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

[72]:
```
# Our solution uses 1 statement split over 3 lines.
fruits = Table().with_columns( #SOLUTION
        "fruit name", make_array("apple", "orange", "pineapple"), #SOLUTION
        "count", make_array(4, 3, 3)) #SOLUTION
fruits
```

[72]:

fruit name	count
apple	4
orange	3
pineapple	3

[73]:
```
_ = ok.grade('q6_1')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
```

```
    Passed: 1
    Failed: 0
[oooooooooo] 100.0% 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`.

[74]:
```
inventory = Table.read_table("inventory.csv") #SOLUTION
inventory
```

[74]:
```
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
```

[75]:
```
_ = ok.grade('q6_2')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooo] 100.0% passed
```

Question 3. Does each box at the fruit stand contain a different fruit?

[76]:
```
# 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" #SOLUTION
all_different
```

[76]:
```
'No'
```

[77]:
```
_ = ok.grade('q6_3')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
```

```
    Passed: 1
    Failed: 0
[oooooooooook] 100.0% 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`.

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

[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

[79]:
```
_ = ok.grade('q6_4')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooook] 100.0% passed
```

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

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

[80]:
```
638
```

[81]:
```
_ = ok.grade('q6_5')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
```

```
    Passed: 1
    Failed: 0
[oooooooooo] 100.0% 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.

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

[83]:
```
106.85
```

[84]:
```
_ = ok.grade('q6_6')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooo] 100.0% 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 count, so that the "count" is the amount of fruit remaining after Saturday.

[85]:
```
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
```

[85]:

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

[86]:
```
_ = ok.grade('q6_7')
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

----------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[ooooooooook] 100.0% passed
```

[ ]: