Homework 2: Arrays and Tables

Please complete this notebook by filling in the cells provided. When you're done:

1. Select `Run All` from the `Cell` menu to ensure that you have executed all cells.
2. Select `Download as PDF via LaTeX (.pdf)` from the `File` menu
3. Read that file! If any of your lines are too long and get cut off, we won't be able to see them, so break them up into multiple lines and download again.
4. Submit that downloaded file to Gradescope.

If you cannot submit online, come to office hours for assistance. The office hours schedule appears on data8.org/fa16/weekly.html (http://data8.org/fa16/weekly.html).

This assignment is due Thursday, September 8 at 5PM. You will receive an early submission bonus point if you turn it in by Wednesday, September 7 at 5PM. Directly sharing answers is not okay, but discussing problems with course staff or with other students is encouraged.

Reading:

- Textbook chapter 4 (http://www.inferentialthinking.com/chapters/04/data-types.html)

Run the cell below to prepare the notebook. **Passing the automatic tests does not guarantee full credit on any question.** The tests are provided to help catch some common errors, but it is *your* responsibility to answer the questions correctly.

In [ ]:
```
# Run this cell to set up the notebook, but please don't change it.
import numpy as np
from datascience import *

from client.api.assignment import load_assignment
tests = load_assignment('hw02.ok')
```

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.

In [2]:
```
# Our solution involved one extra line of code before creating
# weird_numbers.
...
weird_numbers = ...
weird_numbers
```

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

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

In [4]:
```
book_title_words = ...
book_title_words
```

In [5]:
```
_ = tests.grade('q1_2')
```

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 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)`.

In [6]:
```
with_commas = ...
without_commas = ...

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

In [7]:
```
_ = tests.grade('q1_3')
```

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. Unfortunately, this small detail can cause a disproportionate amount of confusion. So it's useful to practice indexing arrays, even if it's not a particularly important concept.

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

In [13]:
```
some_numbers = make_array(-1, -3, -6, -10, -15)

third_element = ...
third_element
```

In [14]:
```
_ = tests.grade('q2_1')
```

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.

(Don't worry about the "Table" stuff -- all you need to do is run the cell and replace the `"???"` expressions with the right strings. You'll learn about tables soon.)

In [18]:
```
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
```

In [19]:
```
_ = tests.grade('q2_2')
```

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?

In [20]:
```
index_of_last_element = ...
```

In [21]:
```
_ = tests.grade('q2_3')
```

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 `length` of that array (an integer).

Question 4. The cell below loads an array called `president_birth_years`. Find the last element in that array. (It's the birth year of John F. Kennedy, as of 2016 the most recently-born deceased US President.) Do this **without** examining the data manually. (So your answer shouldn't include anything that could change if we changed the dataset.)

In [22]:
```
# Don't worry about how this line works -- you'll learn about
# tables soon.
president_birth_years =
Table.read_table("president_births.csv").column('Birth Year')

most_recent_birth_year = ...
most_recent_birth_year
```

In [23]:
```
_ = tests.grade('q2_4')
```

3. 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 (http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1034367/pdf/medhist00183-0026.pdf).)

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.

Why was it important to study this group?

Write your answer here, replacing this text.

Note: Far from disproving Snow's claim, 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.

4. Flipping Coins

As you're walking down Bancroft one afternoon, a shadowy figure steps out of Sather Lane and offers you the following game:

> "First, you pay me $1 to play. Then, I'll flip 4 fair coins. ("Fair" means that there's a 50% chance the coin lands heads and a 50% chance it lands tails, and all the coins are independent of each other.)
>
> "If all of the coins come up heads, then I'll give you $10."

Let's figure out whether this is a good deal. Answering that question formally requires a little bit of decision theory, which we haven't seen yet, but for now we can at least use a computer to think about how the game works.

First, let's think about the outcomes that can happen. We'll order the coins left to right and denote an outcome by a **string**. For example, if we get all heads, we'll denote that outcome by `"HHHH"`. If we get tails first and then 3 heads, we'll denote that by `"THHH"`.

Question 1. Make an array containing any 3 outcomes that can happen, using this notation. (The array should contain 3 strings.) Don't use the examples we've already given!

In [2]:
```
outcomes = ...
```

In [3]:
```
_ = tests.grade('q4_1')
```

A single coin has 2 outcomes, `"H"` and `"T"`. When you add a coin to any set of coins, that *multiplies* the number of potential outcomes by 2, because for each potential outcome there was before, there are now 2 potential outcomes: one where the new coin is heads, and one where the new coin is tails. So if the shadowy figure flips n coins, then there are 2^n potential outcomes.

Question 2. Compute the number of outcomes for 4 coins. **Use the exponentiation (`**`) operator.**

In [4]:
```
num_outcomes_4_coins = ...
```

In [5]:
```
_ = tests.grade('q4_2')
```

You can run the cell below to see all the outcomes for 4 coins that you didn't put in `outcomes`. You can modify `n` to see all the outcomes for different numbers of coins. (Don't make `n` too big, or the cell might crash your Jupyter server!)

In [6]:
```
# This cell uses some techniques you haven't seen yet.  Don't worry abou
t
# reading it, unless you're curious.
n = 4
digit_to_coin = Table().with_columns("digit", make_array("0", "1"), "coi
n", make_array("T", "H"))
def number_to_outcome(num_digits):
    def converter(number):
        """Converts a number to a coin outcome string, based on its bina
ry representation."""
        bit_string = np.array(bin(number).split("b")).item(1).zfill(num_
digits)
        digits = Table().with_columns("index", np.arange(num_digits), "d
igit", np.array(list(bit_string)))
        heads_tails = "".join(digits.join("digit",
digit_to_coin).sort("index").column("coin"))
        return heads_tails
    return converter
# All the potential outcomes:
all_outcomes = Table().with_column("outcome number", np.arange(2**n)).ap
ply(number_to_outcome(n), "outcome number")
# All the potential outcomes, except the ones you named:
remaining_outcomes = np.array(sorted(set(all_outcomes) - set(outcomes)))
remaining_outcomes
```

You should have found that there were 16 possible outcomes. Since 1 of them nets you $9 and the other 15 net you -$1, if you play 16 games, then on average you'll lose $6. So it's probably not a good idea to play the game. But this would change if there were more or fewer coins!

Question 3. Suppose the shadowy figure had used a different number of coins. Compute the number of outcomes for a game with 1 coin, for a game with 2 coins, for a game with 3 coins, and so on, up to 50 coins. Put the results in an array called `num_outcomes`. (So your answer should look like `array([2, 4, 8, 16, ...])`.)

Hint: Don't write an expression and copy it 50 times! Instead, think about the formula for each entry of the array you want, and use array arithmetic to implement that formula.

In [7]:
```
num_outcomes = ...
num_outcomes
```

In [8]:
```
_ = tests.grade('q4_3')
```

5. Tables

Question 1. Suppose you have 4 apples, 3 oranges, and 3 pineapples. (Perhaps you're using Python to solve a high school Algebra problem.) Create a table that contains this information. It should have two columns: "fruit name" and "count". Give it the name `fruits`.

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

In [2]:
```
# Our solution uses 1 statement split over 3 lines for readability.
# Of course, you can complete the question however you like.
fruits = ...
        ...
        ...
fruits
```

In [48]:
```
_ = tests.grade('q5_1')
```

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`.

In [3]:
```
inventory = ...
inventory
```

In [4]:
```
_ = tests.grade('q5_2')
```

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

Write your answer here, replacing this text.

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`.

In [5]:
```
sales = ...
sales
```

In [6]:
```
_ = tests.grade('q5_4')
```

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

In [8]:
```
total_fruits_sold = ...
total_fruits_sold
```

In [9]:
```
_ = tests.grade('q5_5')
```

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.

In [10]:
```
total_revenue = ...
total_revenue
```

In [11]:
```
_ = tests.grade('q5_6')
```

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.

In [14]:
```
remaining_inventory = ...
    ...
    ...
    ...
remaining_inventory
```

In [13]:
```
_ = tests.grade('q5_7')
```

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