[Index | Exercise 2.1 | Exercise 2.3]

Exercise 2.2

Objectives:

- Work with various containers
- List/Set/Dict Comprehensions
- Collections module
- Data analysis challenge

Most Python programmers are generally familiar with lists, dictionaries, tuples, and other basic datatypes. In this exercise, we'll put that knowledge to work to solve various data analysis problems.

(a) Preliminaries

To get started, let's review some basics with a slightly simpler dataset-- a portfolio of stock holdings. Create a file `readport.py` and put this code in it:

```
# readport.py

import csv

# A function that reads a file into a list of dicts
def read_portfolio(filename):
    portfolio = []
    with open(filename) as f:
        rows = csv.reader(f)
        headers = next(rows)
        for row in rows:
            record = {
                'name' : row[0],
                'shares' : int(row[1]),
                'price' : float(row[2])
            }
            portfolio.append(record)
    return portfolio
```

This file reads some simple stock market data in the file `Data/portfolio.csv`. Use the function to read the file and look at the results:

```
>>> portfolio = read_portfolio('Data/portfolio.csv')
>>> from pprint import pprint
>>> pprint(portfolio)
[{'name': 'AA', 'price': 32.2, 'shares': 100},
 {'name': 'IBM', 'price': 91.1, 'shares': 50},
 {'name': 'CAT', 'price': 83.44, 'shares': 150},
```

```
 {'name': 'MSFT', 'price': 51.23, 'shares': 200},
 {'name': 'GE', 'price': 40.37, 'shares': 95},
 {'name': 'MSFT', 'price': 65.1, 'shares': 50},
 {'name': 'IBM', 'price': 70.44, 'shares': 100}]
>>>
```

In this data, each row consists of a stock name, a number of held shares, and a purchase price. There are multiple entries for certain stock names such as MSFT and IBM.

(b) Comprehensions

List, set, and dictionary comprehensions can be a useful tool for manipulating data. For example, try these operations:

```
>>> # Find all holdings more than 100 shares
>>> [s for s in portfolio if s['shares'] > 100]
[{'name': 'CAT', 'shares': 150, 'price': 83.44},
 {'name': 'MSFT', 'shares': 200, 'price': 51.23}]

>>> # Compute total cost (shares * price)
>>> sum([s['shares']*s['price'] for s in portfolio])
44671.15
>>>

>>> # Find all unique stock names (set)
>>> { s['name'] for s in portfolio }
{'MSFT', 'IBM', 'AA', 'GE', 'CAT'}
>>>

>>> # Count the total shares of each of stock
>>> totals = { s['name']: 0 for s in portfolio }
>>> for s in portfolio:
        totals[s['name']] += s['shares']

>>> totals
{'AA': 100, 'IBM': 150, 'CAT': 150, 'MSFT': 250, 'GE': 95}
>>>
```

(c) Collections

The `collections` module has a variety of classes for more specialized data manipulation. For example, the last example could be solved with a `Counter` like this:

```
>>> from collections import Counter
>>> totals = Counter()
>>> for s in portfolio:
        totals[s['name']] += s['shares']
```

```
>>> totals
Counter({'MSFT': 250, 'IBM': 150, 'CAT': 150, 'AA': 100, 'GE': 95})
>>>
```

Counters are interesting in that they support other kinds of operations such as ranking and mathematics. For example:

```
>>> # Get the two most common holdings
>>> totals.most_common(2)
[('MSFT', 250), ('IBM', 150)]
>>>

>>> # Adding counters together
>>> more = Counter()
>>> more['IBM'] = 75
>>> more['AA'] = 200
>>> more['ACME'] = 30
>>> more
Counter({'AA': 200, 'IBM': 75, 'ACME': 30})
>>> totals
Counter({'MSFT': 250, 'IBM': 150, 'CAT': 150, 'AA': 100, 'GE': 95})
>>> totals + more
Counter({'AA': 300, 'MSFT': 250, 'IBM': 225, 'CAT': 150, 'GE': 95, 'ACME':
30})
>>>
```

The `defaultdict` object can be used to group data. For example, suppose you want to make it easy to find all matching entries for a given name such as IBM. Try this:

```
>>> from collections import defaultdict
>>> byname = defaultdict(list)
>>> for s in portfolio:
        byname[s['name']].append(s)

>>> byname['IBM']
[{'name': 'IBM', 'shares': 50, 'price': 91.1}, {'name': 'IBM', 'shares':
100, 'price': 70.44}]
>>> byname['AA']
[{'name': 'AA', 'shares': 100, 'price': 32.2}]
>>>
```

The key feature that makes this work is that a defaultdict automatically initializes elements for you--allowing an insertion of a new element and an `append()` operation to be combined together.

(c) Data Analysis Challenge

In the last exercise you just wrote some code to read CSV-data related to the Chicago Transit Authority. For example, you can grab the data as dictionaries like this:

```
>>> import readrides
>>> rows = readrides.read_rides_as_dicts('Data/ctabus.csv')
>>>
```

It would be a shame to do all of that work and then do nothing with the data.

In this exercise, you task is this: write a program to answer the following three questions:

1. How many bus routes exist in Chicago?
2. How many people rode the number 22 bus on February 2, 2011? What about any route on any date of your choosing?
3. What is the total number of rides taken on each bus route?
4. What five bus routes had the greatest ten-year increase in ridership from 2001 to 2011?

You are free to use any technique whatsoever to answer the above questions as long as it's part of the Python standard library (i.e., built-in datatypes, standard library modules, etc.).

[Solution | Index | Exercise 2.1 | Exercise 2.3]
