Programming Project 7

(Updated Oct 23 with more explanations for dictionary operations; Oct. 24 allowed flexibility in what merge_dict returns. Oct. 25 information was added about setting up for plotting and testing with smaller test files.)

This assignment is worth 50 points (5.0% of the course grade) and must be completed and turned in before 11:59 on Monday, October 31, 2016.

Assignment Overview

This assignment will give you more experience on the use of:

1. Dictionaries
2. Functions
3. Plotting with pylab

The goal of this project is to gain practice with dictionaries and functions, as well as plotting using the pylab module.

Process files from the Environmental Protection Agency (EPA) to extract fuel economy data for the various manufacturers. Then, plot the overall average city and overall average highway fuel economy data for four manufacturers: Ford, GM, Honda, and Toyota.

Assignment Deliverable

The deliverable for this assignment is the following file:

`proj07.py` – the source code for your Python program

Be sure to use the specified file name and to submit it for grading via the **handin system** before the project deadline.

Background

Vehicle fuel economy has been something of varying importance to consumers over the years – when gas prices are low, consumers tend not to focus as much on fuel efficiency, but when gas prices are high, they tend to buy more efficient vehicles. Additionally, there might be a difference in philosophy between American manufacturers and those from Asia, especially when marketing to Americans. Your goal is to plot data from two American automakers (Ford & GM) and two Asian automakers (Honda & Toyota) to see these trends and differences.

For the purposes of this assignment:

Ford = Ford + Mercury + Lincoln
GM = Chevrolet + Pontiac + Buick + GMC + Cadillac + Oldsmobile + Saturn
Honda = Honda + Acura
Toyota = Toyota + Lexus + Scion

Project Description / Specification

1) Four files of vehicle data from the EPA have been provided for the four decades spanning 1980-2015: 1980s.csv, 1990s.csv, 2000s.csv, and 2010s.csv. You can examine these files by opening them in Excel, Notepad, Textedit, or similar text editors. Data in each file is delimited by commas. Note that each file has a header describing the contents of the columns. Of particular interest to us are the manufacturer (column 46), year (63), city fuel economy (4), and highway fuel economy (34). Ignore any data from the 2017 model year since it is incomplete.

2) You must implement the following functions:

a) `open_files()`

In this function, you are required to prompt the user for a one or more decades (e.g. 1980) separated by commas and attempt to open all corresponding files (e.g. 1980s.csv) and add it to the list of opened file pointers. If an error is encountered, inform the user, close all open files and request decades again. There are two types of errors to catch: (1) The only decades supported are 1980, 1990, 2000, 2010 so anything else specified generates an "Error in decade" message, (2) a FileNotFoundError generates a "file not found" error message. Return a list of file pointers.

b) `read_file(input_file)`

In this function, you are required to read the file of data and construct and return a dictionary from that data, and return a list of year. The keys for the dictionary will be the manufacturers and the values will be a collection of dictionaries where the keys are the years. You must use a dictionary of dictionaries, but there are two ways to handle the values of the innermost dictionary. You may choose either (I chose the second, but you may find your recent experience with lists makes the first easier for you). Remember to skip year 2017.

I. **a list of two lists**: a list of the city mileage of vehicles and a list of highway mileage (for that manufacturer and year).

```
{manufacturer: {year: [[],[]]}}
```

That is, a partial sample of the Ford dictionary generated from the file 1980s.csv would look like following with city mileage at index 0 and highway at index 1:

```
{'Ford': {'1984': [[35,…],[47,…]]}}
```

For example with a dictionary named D, to append a new city mileage `m1` to your list:

```
D[manufacturer][year][0].append(m1)
```

Remember that when a year is first encountered you must initialize that list with city mileage `m1` and highway mileage `m2`

```
D[manufacturer][year]=[[m1],[m2]]
```

And, if the manufacturer doesn't exist yet, you must initialize with a year

```
D[manufacturer] = {year:[[m1],[m2]]}
```

II. **a dictionary with two keys**, 'city' and 'highway', with each value a list of mileage.

```
{manufacturer: {year: {'city': [], 'highway': []}}
```

That is, a partial sample of the Ford dictionary generated from the file 1980s.csv would look like

```
{'Ford': {'1984': {'city':[35,…], 'highway':[47,…]}}}
```

For example with a dictionary named D, to append a new city mileage `m1` to your list:

```
D[manufacturer][year]['city'].append(m1)
```

Remember that when a year is first encountered you must initialize that entry with

```
D[manufacturer][year]={'city':[m1],'highway':[m2]}
```

And, if the manufacturer doesn't exist yet, you must initialize with a year

```
D[manufacturer] = {year:{'city':[m1],'highway':[m2]}}
```

In summary there are three cases to consider (illustrated with a list of lists – a dictionary of lists is similar. (Because this is done multiple times I put created a function.)

1. The manufacturer is not yet a key: initialize with a year such as
   `D[manufacturer] = {year:[[m1],[m2]]}`
2. The manufacturer exists as a key, but this year does not yet exist: add an entry for the year
   `D[manufacturer][year]=[[m1],[m2]]`
3. Both the manufacturer and year exist: append a new value to the existing list
   `D[manufacturer][year][0].append(m1)`

How do you check if a manufacturer or year is a key? Use "in" as in

```
if manufacturer in D:
```

when checking year you do the following

```
if year in D[manufacturer]:
```

c) `merge_dict(target, source)`

In this function, the target dictionary is updated with the contents of the source dictionary. You have the option to return the updated dictionary or since a dictionary is mutable simply change the dictionary in the function and not return anything. For example, if a `manufacturer` is already in the target,

```
target[manufacturer].update(source[manufacturer])
```

In our case what this does is add the data for the years in the source to the target. As usual, if the `manufacturer` isn't yet in the target you will need to initialize with

```
target[manufacturer] = source[manufacturer]
```

Here is a session showing how update works:

```
In [22]: target = {'Ford':{1992:[1,2,3]}}

In [23]: source = {'Ford':{2000:[6,7,8]}}

In [24]: target['Ford'].update(source['Ford'])

In [25]: target
Out[25]: {'Ford': {1992: [1, 2, 3], 2000: [6, 7, 8]}}
```

Hint: don't overthink this function it is a simple 6-line function.

d) `plot_mileage(years,city,highway)` We provide this function. It will plot the city and highway mileage data, each in their own plot. Input: years, a list of years; city, a dictionary with manufacturer as key and list of annual mileage as value; highway, a similar dictionary with a list of highway mileage as values. Note that in order to plot by year the lists must be sorted by year. I did this by creating a list of tuples with `(year, avg-mileage)`, sorting that list, and then extracting a list of avg-mileage in order. Requirement: *all lists must be the same length*. The dictionary structure for `city` is `{manufacturer: []}`. `highway` is similar. For example,

```
{'Ford':[1,3,5,7], 'GM': [2,4,6,8]}
```

On some inputs the x-axis labels may not print nicely. That is not your problem—I will update with a more robust plotting function if I can.

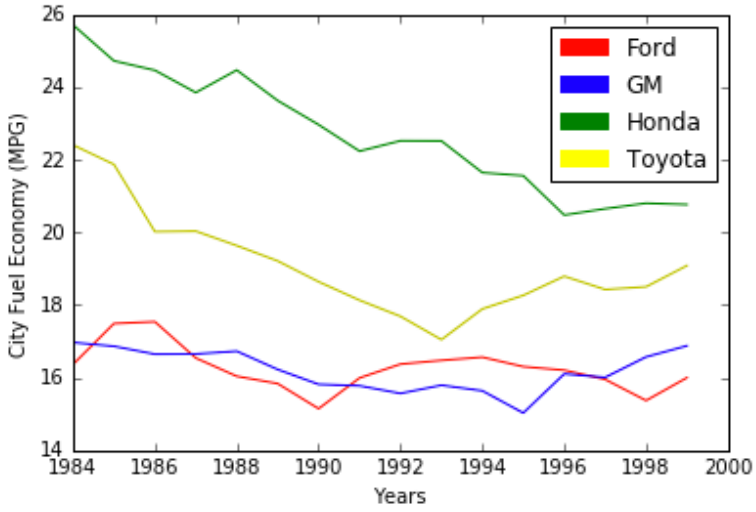
3) After plotting the data calculate the overall average for city mileage and highway mileage for each manufacturer over the specified decades. For example, for Ford take the list of city values plotted

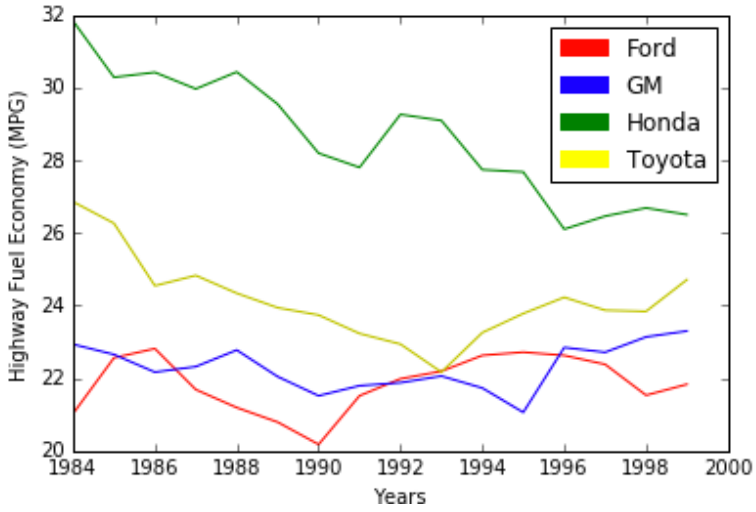
and calculate their average. Note that calculating the average of averages results in a meaningless statistic, but it provides a way for you to check your data.

4) You may want to use csv.reader, but you are not required to.
5) You may use other functions. For example, I had a function to add data to my dictionary and a function to convert data into the format of the plot routine.
6) Do not loop to ask for another plot.

Sample Output

```
Input multiple decades separated by commas, e.g. 1980, 1990, 2000:
1980,1990
```






```
Manufactures' average for 1980, 1990
City
   Company: Mileage
         GM: 16.20
      Honda: 22.68
       Ford: 16.26
     Toyota: 19.10
Highway
   Company: Mileage
         GM: 22.31
      Honda: 28.62
       Ford: 21.85
     Toyota: 24.16
```

Testing with smaller test files: 1980s_small.csv, 1990s_small.csv

We have provided two smaller test files so you can test your code (the actual files have more than 6000 lines each). The first test file, 1980s_small.csv, has 20 lines of data with Honda and Toyota vehicles, but no GM or Ford (there are some other vehicles so you can test that your code ignores them). Data in this file comes from only two years: 1984 and 1986. I printed the main dictionary after each phase of the program. The first set of output comes from reading the 1980s data. The second set uses two files, from the 1980s and 1990s. The 1990s small file has 39 lines (and has no Ford or GM data). It does add Acura and Lexus so you can test that part of your program. Also, with two files you get to test the merging of the files. Neither set of output includes graphing, but I do display the dictionaries that are passed to the plotting routine.

```
Input multiple decades separated by commas, e.g. 1980, 1990, 2000: 1980
TESTING: opening file 1980s_small.csv

Dictionary after read_file:
{'Honda': {1984: {'city': [38, 36, 32], 'hwy': [47, 45, 39]}, 1986:
{'city': [24, 21, 22], 'hwy': [29, 29, 28]}}, 'Toyota': {1984: {'city':
[33, 32], 'hwy': [40, 38]}, 1986: {'city': [24, 23, 23, 26], 'hwy': [28,
29, 29, 33]}}}

Dictionary after reading all files:
{'Honda': {1984: {'city': [38, 36, 32], 'hwy': [47, 45, 39]}, 1986:
{'city': [24, 21, 22], 'hwy': [29, 29, 28]}}, 'Toyota': {1984: {'city':
[33, 32], 'hwy': [40, 38]}, 1986: {'city': [24, 23, 23, 26], 'hwy': [28,
29, 29, 33]}}}

Dictionary after calculating averages:
{'Honda': {1984: {'city': 35.333333333333336, 'hwy': 43.666666666666664},
1986: {'city': 22.333333333333332, 'hwy': 28.666666666666668}}, 'Toyota':
{1984: {'city': 32.5, 'hwy': 39.0}, 1986: {'city': 24.0, 'hwy': 29.75}}}

Dictionaries for plotting:
city:
{'Honda': [35.333333333333336, 22.333333333333332], 'Toyota': [32.5, 24.0]}
highway
{'Honda': [43.666666666666664, 28.666666666666668], 'Toyota': [39.0,
29.75]}

Manufactures' average for decades 1980

City
   Company: Mileage
     Honda: 28.83
    Toyota: 28.25
Highway
   Company: Mileage
     Honda: 36.17
    Toyota: 34.38

========================================================================

Input multiple decades separated by commas, e.g. 1980, 1990, 2000: 1980,
1990
TESTING: opening file 1980s_small.csv
TESTING: opening file 1990s_small.csv

Dictionary after read_file:
```

{'Honda': {1984: {'city': [38, 36, 32], 'hwy': [47, 45, 39]}, 1986: {'city': [24, 21, 22], 'hwy': [29, 29, 28]}}, 'Toyota': {1984: {'city': [33, 32], 'hwy': [40, 38]}, 1986: {'city': [24, 23, 23, 26], 'hwy': [28, 29, 29, 33]}}}

Dictionary after read_file:

{'Honda': {1992: {'city': [39, 36, 35, 16, 17, 17, 16], 'hwy': [49, 46, 43, 22, 22, 22, 24]}, 1990: {'city': [40, 36, 21, 17], 'hwy': [47, 44, 26, 22]}}, 'Toyota': {1992: {'city': [27, 17, 16, 16, 16], 'hwy': [33, 24, 20, 21, 20]}, 1990: {'city': [27, 26, 17, 17], 'hwy': [33, 33, 23, 24]}, 1991: {'city': [11, 10], 'hwy': [13, 10]}}}

Dictionary after reading all files:

{'Honda': {1984: {'city': [38, 36, 32], 'hwy': [47, 45, 39]}, 1992: {'city': [39, 36, 35, 16, 17, 17, 16], 'hwy': [49, 46, 43, 22, 22, 22, 24]}, 1986: {'city': [24, 21, 22], 'hwy': [29, 29, 28]}, 1990: {'city': [40, 36, 21, 17], 'hwy': [47, 44, 26, 22]}}, 'Toyota': {1984: {'city': [33, 32], 'hwy': [40, 38]}, 1992: {'city': [27, 17, 16, 16, 16], 'hwy': [33, 24, 20, 21, 20]}, 1986: {'city': [24, 23, 23, 26], 'hwy': [28, 29, 29, 33]}, 1990: {'city': [27, 26, 17, 17], 'hwy': [33, 33, 23, 24]}, 1991: {'city': [11, 10], 'hwy': [13, 10]}}}

Dictionary after calculating averages:

{'Honda': {1984: {'city': 35.333333333333336, 'hwy': 43.666666666666664}, 1992: {'city': 25.142857142857142, 'hwy': 32.57142857142857}, 1986: {'city': 22.333333333333332, 'hwy': 28.666666666666668}, 1990: {'city': 28.5, 'hwy': 34.75}}, 'Toyota': {1984: {'city': 32.5, 'hwy': 39.0}, 1992: {'city': 18.4, 'hwy': 23.6}, 1986: {'city': 24.0, 'hwy': 29.75}, 1990: {'city': 21.75, 'hwy': 28.25}, 1991: {'city': 10.5, 'hwy': 11.5}}}

Dictionaries for plotting:

city:

{'Honda': [35.333333333333336, 22.333333333333332, 28.5, 25.142857142857142], 'Toyota': [32.5, 24.0, 21.75, 10.5, 18.4]}

highway

{'Honda': [43.666666666666664, 28.666666666666668, 34.75, 32.57142857142857], 'Toyota': [39.0, 29.75, 28.25, 11.5, 23.6]}

Manufactures' average for decades 1980, 1990

City
   Company: Mileage
     Honda: 27.83
    Toyota: 21.43
Highway
   Company: Mileage
     Honda: 34.91
    Toyota: 26.42

==========

Educational Research

When you have completed the project insert the 5-line comment specified below.

For each of the following statements, please respond with how much they apply to your experience completing the programming project, on the following scale:

1 = Strongly disagree / Not true of me at all
2
3
4 = Neither agree nor disagree / Somewhat true of me
5
6
7 = Strongly agree / Extremely true of me

*********Please note that your responses to these questions will not affect your project grade, so please answer as honestly as possible*.***

Q1: Upon completing the project, I felt proud/accomplished

Q2: While working on the project, I often felt frustrated/annoyed

Q3: While working on the project, I felt inadequate/stupid

Q4: Considering the difficulty of this course, the teacher, and my skills, I think I will do well in this course.

Please insert your answers into the bottom of your project program as a comment, formatted exactly as follows (so we can write a program to extract them).

```
# Questions
# Q1: 5
# Q2: 3
# Q3: 4
# Q4: 6
```