Dalziel2016_solution

December 28, 2018

1 Solution of 3.8.1, Dalziel *et al.* 2016

1.1 Write a program that extracts the names of all the cities in the database (one city per entry)

For this exercise, we need to a) open the file `data/Dalziel2016_data.csv` for reading; b) read all the lines; c) add the name of the city to a data structure, making sure that we have no repeated entry. For this reason, we're going to work with sets. We start by initializing an empty set called `cities`:

```
In [1]: cities = set([]) # initialize an empty set
```

Now we open the file for reading. We use the `with` statement that takes care of closing the file:

```
In [2]: import csv # we this module to handle csv files
        with open('../data/Dalziel2016_data.csv', 'r') as f: # 'r' stands for reading
            my_csv = csv.DictReader(f) # set up the csv reader
            for line in my_csv: # loop over all lines
                print(line)
                break # break the loop after printing the first line to inspect results
```

```
OrderedDict([('biweek', '1'), ('year', '1906'), ('loc', 'BALTIMORE'), ('cases', 'NA'), ('pop',
```

In the code above, we have imported the module `csv`, which allows us to parse character-delimited files. In this case, we do not need to specify any special option, as we're reading a plain-vanilla `csv` file, delimited by commas.

Having opened the file, we create a `DictReader` object, which parses each line, creating a dictionary whose entries are the values for each of the columns (named as specified by the *header* of the `csv` file).

You can see that in the dictionary, the city is identified by the *key* `'loc'`. We can therefore add the *value* `line['loc']` to the set, completing the exercise:

```
In [3]: import csv # we use the csv module, as we want to read a csv file
        with open('../data/Dalziel2016_data.csv', 'r') as f: # 'r' stands for reading
            my_csv = csv.DictReader(f)
            for line in my_csv:
                cities.add(line['loc'])
```

Now all the cities are stored in the set `cities`, with all the duplicates automatically removed (as we're using a set):

```
In [4]: cities
```

```
Out[4]: {'BALTIMORE',
         'BOSTON',
         'BRIDGEPORT',
         'BUFFALO',
         'CHICAGO',
         'CINCINNATI',
         'CLEVELAND',
         'COLUMBUS',
         'DENVER',
         'DETROIT',
         'DULUTH',
         'FALL RIVER',
         'GRAND RAPIDS',
         'HARTFORD',
         'INDIANAPOLIS',
         'KANSAS CITY',
         'LOS ANGELES',
         'MILWAUKEE',
         'MINNEAPOLIS',
         'NASHVILLE',
         'NEW HAVEN',
         'NEW ORLEANS',
         'NEW YORK',
         'NEWARK',
         'PHILADELPHIA',
         'PITTSBURGH',
         'PROVIDENCE',
         'READING.US',
         'RICHMOND',
         'ROCHESTER',
         'SALT LAKE CITY',
         'SAN FRANCISCO',
         'SEATTLE',
         'SPOKANE',
         'SPRINGFIELD',
         'ST LOUIS',
         'TOLEDO',
         'TRENTON',
         'WASHINGTON',
         'WORCESTER'}
```

1.2 Write a program that creates a dictionary where the keys are the cities, and the values are the number of records (rows) for that city in the data.

This task requires a slightly different approach. We need to keep track of how many records are associated with each city. We can therefore create a dictionary `citycount` storing the city (*key*) and the associated number of records (*value*).

Because initially the dictionary is empty, every time we encounter a new city we need to add a *key* to the dictionary. The simplest way to do this is to use the dictionary method `get`, which allows us to either update the value (if the key is already present), or to add a new key (if the key is not present). For example:

```
In [5]: a = {} # an empty dictionary
        a['my_new_key'] = a.get('my_new_key', 0) + 1
        a

Out[5]: {'my_new_key': 1}
```

The code above shows that when the key is not already, present, the key will be added, and its value will be initially `1`. If on the other hand the key is present, we will simply increment its associated value:

```
In [6]: a['my_new_key'] = a.get('my_new_key', 0) + 1
        a

Out[6]: {'my_new_key': 2}
```

With this at hand, we can write our program:

```
In [7]: citycount = {} # initiate an empty dictionary
        import csv # we use the csv module, as we want to read a csv file
        with open('../data/Dalziel2016_data.csv', 'r') as f: # 'r' stands for reading
            my_csv = csv.DictReader(f)
            for line in my_csv:
                # this is the city to update
                mycity = line['loc']
                # if it's present, increment the value
                # if it's not present, initialize to 1
                citycount[mycity] = citycount.get(mycity, 0)
                citycount[mycity] = citycount[mycity] + 1
```

That's it. Let's print the counts for a few cities:

```
In [8]: for city in ['CHICAGO', 'LOS ANGELES', 'NEW YORK']:
            print(city, citycount[city])

CHICAGO 1118
LOS ANGELES 1118
NEW YORK 1118
```

1.3 Write a program that calculates the mean population for each city, obtained by averaging the value of pop.

We can proceed as before. Remember that the mean is the sum of elements divided by the number of elements ($\mathbb{E}[x_1, x_2, x_3, x_4, \ldots, x_n] = \frac{1}{n}\sum_{i=1}^{n} x_i$).

Therefore, we can simply keep summing the population at each step, and at the end divide by the number of records. We create a new dictionary, `citypop` whose value is a *list*, containing the current sum of the population, and the number of records for the city:

```
In [9]: citypop = {}
        import csv # we use the csv module, as we want to read a csv file
        with open('../data/Dalziel2016_data.csv', 'r') as f: # 'r' stands for reading
            my_csv = csv.DictReader(f)
            for line in my_csv:
                # this is the city to update
                mycity = line['loc']
                # current pop
                pop = float(line['pop']) # transform to float
                # if it's present, increment the value
                # if it's not present, initialize a list with both population and count as zer
                citypop[mycity] = citypop.get(mycity, [0,0])
                # update population (stored as first value of list)
                citypop[mycity][0] = citypop[mycity][0] + pop
                # update number of records (stored as second value of list)
                citypop[mycity][1] = citypop[mycity][1] + 1
```

```
In [10]: citypop
```

```
Out[10]: {'BALTIMORE': [852064394.4319992, 1118],
          'BOSTON': [838182525.1315998, 1118],
          'BRIDGEPORT': [153992147.5000699, 1118],
          'BUFFALO': [590188826.5568998, 1118],
          'CHICAGO': [3346478160.099001, 1118],
          'CINCINNATI': [476570324.3801995, 1118],
          'CLEVELAND': [895654069.0593997, 1118],
          'COLUMBUS': [296369301.20559984, 1118],
          'DENVER': [316218012.39949924, 1118],
          'DETROIT': [1386865097.4254, 1118],
          'DULUTH': [108107768.67043993, 1118],
          'FALL RIVER': [131021870.39960006, 1118],
          'GRAND RAPIDS': [166710967.5439999, 1118],
          'HARTFORD': [163331387.46438012, 1118],
          'INDIANAPOLIS': [375166935.70720017, 1118],
          'KANSAS CITY': [259178329.18899986, 1118],
          'LOS ANGELES': [1130219693.2247996, 1118],
          'MILWAUKEE': [573871553.6460003, 1118],
          'MINNEAPOLIS': [472055535.7656998, 1118],
          'NASHVILLE': [158006495.8098898, 1118],
          'NEW HAVEN': [174722096.8123, 1118],
          'NEW ORLEANS': [485409089.0958999, 1118],
          'NEW YORK': [7128667329.930001, 1118],
          'NEWARK': [461290195.0237001, 1118],
          'PHILADELPHIA': [2059222693.4399996, 1118],
          'PITTSBURGH': [696554638.3960005, 1118],
          'PROVIDENCE': [271998881.5344998, 1118],
          'READING.US': [119528247.67913005, 1118],
          'RICHMOND': [195617340.76740003, 1118],
          'ROCHESTER': [332343203.71729976, 1118],
          'SALT LAKE CITY': [146091306.56043985, 1118],
          'SAN FRANCISCO': [637121635.7180995, 1118],
          'SEATTLE': [374078074.35140014, 1118],
          'SPOKANE': [127242978.51299988, 1118],
          'SPRINGFIELD': [148986059.7662199, 1118],
          'ST LOUIS': [876436587.4204999, 1118],
          'TOLEDO': [282808550.9529003, 1118],
          'TRENTON': [131378128.07106006, 1118],
          'WASHINGTON': [572394549.4401004, 1118],
          'WORCESTER': [203141985.43980017, 1118]}
```

Excellent. Now each key in the dictionary indexes a list whose first element is the sum of all the population values, and the second element is the number of records that contributed to the sum. To obtain the average population, we divide the first by the second:

```
In [11]: for city in citypop.keys():
             citypop[city][0] = citypop[city][0] / citypop[city][1]
```

Let's see some of the averages to make sure they make sense:

```
In [12]: for city in ['CHICAGO', 'LOS ANGELES', 'NEW YORK']:
             print(city, citypop[city][0])
```

```
CHICAGO 2993272.0573336324
LOS ANGELES 1010929.958161717
NEW YORK 6376267.736967801
```

If we want print only a few decimals, we can use `round`:

```
In [13]: for city in ['CHICAGO', 'LOS ANGELES', 'NEW YORK']:
             print(city, round(citypop[city][0],1))
```

```
CHICAGO 2993272.1
LOS ANGELES 1010930.0
NEW YORK 6376267.7
```

1.4 Write a program that calculates the mean population for each city and year.

Though this exercise looks very much like the previous one, we need to change the data structure slightly. In fact, now each city contains many years, and each year should index the corresponding population. The following solution uses a dictionary (where the keys are the cities) of dictionaries (where the keys are the years) of lists (accumulated population, number of records per year)!

```
In [14]: cityyear = {}
```

```
In [15]: cityyear = {}
         import csv # we use the csv module, as we want to read a csv file
         with open('../data/Dalziel2016_data.csv', 'r') as f: # 'r' stands for reading
             my_csv = csv.DictReader(f)
             for line in my_csv:
                 # this is the city to update
                 mycity = line['loc']
                 # this is the year to update
                 year = line['year']
                 # current pop
                 pop = float(line['pop']) # transform to float
                 # make sure the city is in the dictionary, or initialize
                 cityyear[mycity] = cityyear.get(mycity, {})
                 # make sure the year is in the sub-dictionary, or initialize
                 cityyear[mycity][year] = cityyear[mycity].get(year, [0,0])
                 # now proceed as for exercise 3 but access the inner dictionary
                 # update population
                 cityyear[mycity][year][0] = cityyear[mycity][year][0] + pop
                 # update number of records
                 cityyear[mycity][year][1] = cityyear[mycity][year][1] + 1
         # now compute averages
         for city in cityyear.keys():
             for year in cityyear[city].keys():
                 cityyear[city][year][0] = cityyear[city][year][0] / cityyear[city][year][1]
```

Let's look at the results for Chicago: you can see that the population grew by more than 50% in the period covered by the data!

```
In [16]: # a dictionary has no natural order but here we want to order by year
         # store the years in a list
         years = list(cityyear['CHICAGO'].keys())
         # sort the years
         years.sort() # this is done in place!
         # now print population for each year
         for year in years:
             print(year, round(cityyear['CHICAGO'][year][0]))
```

```
1906 2024453
1907 2071166
1908 2117046
```

1909	2162137
1910	2206507
1911	2250497
1912	2294736
1913	2339879
1914	2386581
1915	2435498
1916	2487285
1917	2542598
1918	2602091
1919	2666421
1920	2736144
1921	2810674
1922	2888232
1923	2966923
1924	3044854
1925	3120130
1926	3190859
1927	3255145
1928	3311095
1929	3356815
1930	3390564
1931	3412364
1932	3424085
1933	3427769
1934	3425462
1935	3419208
1936	3411050
1937	3403033
1938	3397202
1939	3395600
1940	3400149
1941	3411346
1942	3428198
1943	3449572
1944	3474333
1945	3501348
1946	3529481
1947	3557600
1948	3584569