Exemplar_ Dictionaries & sets

January 7, 2024

Exemplar: Dictionaries and sets

0.1 Introduction

In this lab, you will practice creating, modifying, and working with data structures in Python. This will develop your knowledge of different kinds of data structures and the different operations that you can perform with them to answer questions about the data. This will help you prepare for projects you may encounter where you will need to use data structures to store and keep track of data.

In your work as an analyst, you are continuing your research into air quality data collected by the U.S. Environmental Protection Agency (EPA). The air quality index (AQI) is a number that runs from 0 to 500. The higher the AQI value, the greater the level of air pollution and the greater the health concern. For example, an AQI value of 50 or below represents good air quality, while an AQI value over 300 represents hazardous air quality. Refer to this guide from AirNow.gov for more information.

In this activity, you will create, modify, and update dictionaries and sets. You will also be working with more data than in previous labs to more closely resemble situations encountered by working data professionals.

0.2 Tips for completing this lab

As you navigate this lab, keep the following tips in mind:

- `### YOUR CODE HERE ###` indicates where you should write code. Be sure to replace this with your own code before running the code cell.
- Feel free to open the hints for additional guidance as you work on each task.
- To enter your answer to a question, double-click the markdown cell to edit. Be sure to replace the "[Double-click to enter your responses here.]" with your own answer.
- You can save your work manually by clicking File and then Save in the menu bar at the top of the notebook.
- You can download your work locally by clicking File and then Download and then specifying your preferred file format in the menu bar at the top of the notebook.

Task 1: Create a dictionary to store information

Dictionaries are useful when you need a data structure to store information that can be referenced or looked up.

In this task you'll begin with three `list` objects:

- `state_list` - an ordered list of the state where each data point was recorded
- `county_list` - an ordered list of the county where each data point was recorded
- `aqi_list` - an ordered list of AQI records

As a refresher, here is an example table of some of the information contained in these variables:

state_name	county_name	aqi
Arizona	Maricopa	9
California	Alameda	11
California	Sacramento	35
Kentucky	Jefferson	6
Louisiana	East Baton Rouge	5

Reminder: This lab uses more data than the last one. This table is just a small sample of the information contained in the three lists that are provided for this activity.

0.3 1a: Create a list of tuples

Begin with an intermediary step to prepare the information to be put in a dictionary.

- Convert `state_list`, `county_list`, and `aqi_list` to a list of tuples, where each tuple contains information for a single record: `(state, county, aqi)`.
- Assign the result to a variable called `epa_tuples`.

[1]:
```
# RUN THIS CELL TO IMPORT YOUR DATA
import ada_c2_labs as lab
state_list = lab.fetch_epa('state')
county_list = lab.fetch_epa('county')
aqi_list = lab.fetch_epa('aqi')
```

[2]:
```
### YOUR CODE HERE ###
epa_tuples = list(zip(state_list, county_list, aqi_list))
```

Hint 1

Refer to what you learned about element-wise combination of iterable objects.

Hint 2

It's possible to create a list of tuples using a loop, but there's a much simpler way of accomplishing the task using a built-in Python function.

Hint 3

The `zip()` function accepts any number of iterable objects as arguments. If the arguments are all of equal length, the function returns an iterator object of tuples, where each tuple contains `element[i]` of each argument.

You can then either loop over the iterator object or pass it to the `list()` function to unpack its values.

Refer to the zip() Python documentation for more information.

0.4 1b: Create a dictionary

Now that you have a list of tuples containing AQI records, use it to create a dictionary that allows you to look up a state and get all the county-AQI pairs associated with that state.

- Create a dictionary called `aqi_dict`:
  - Use a loop to unpack information from each tuple in `epa_tuples`.
  - Your dictionary's keys should be states.
  - The value at each key should be a list of tuples, where each tuple is a county-AQI pair of a record from a given state.

Example:

```
[IN]  aqi_dict['Vermont']
[OUT] [('Chittenden', 18.0),
       ('Chittenden', 20.0),
       ('Chittenden', 3.0),
       ('Chittenden', 49.0),
       ('Rutland', 15.0),
       ('Chittenden', 3.0),
       ('Chittenden', 6.0),
       ('Rutland', 3.0),
       ('Rutland', 6.0),
       ('Chittenden', 5.0),
       ('Chittenden', 2.0)]
```

[3]:
```
### YOUR CODE HERE ###
aqi_dict = {}
for state, county, aqi in epa_tuples:
    if state in aqi_dict:
        aqi_dict[state].append((county, aqi))
    else:
        aqi_dict[state] = [(county, aqi)]
```

[4]:
```
aqi_dict['Vermont']
```

[4]:
```
[('Chittenden', 18.0),
 ('Chittenden', 20.0),
 ('Chittenden', 3.0),
 ('Chittenden', 49.0),
 ('Rutland', 15.0),
 ('Chittenden', 3.0),
 ('Chittenden', 6.0),
 ('Rutland', 3.0),
```

```
 ('Rutland', 6.0),
 ('Chittenden', 5.0),
 ('Chittenden', 2.0)]
```

Hint 1

Refer to what you learned about unpacking tuples in loops and creating dictionaries.

Hint 2

There are 3 elements in each tuple in `epa_tuples`. Therefore, for each iteration of your loop, you'll need to unpack these 3 elements into their own variables.

Hint 3

With each loop iteration, check whether the state is already a key in `aqi_dict`. Then, use conditional logic: * If it is, append the county and AQI as a tuple to the list at that state's value. * If it's not, assign the state as a new key in `aqi_dict`, with a value that is a list containing the county-AQI tuple.

1 Task 2: Use the dictionary to retrieve information

Now that you have a dictionary of county-AQI readings by state, you can use it to retrieve information and draw further insight from your data.

1.1 2a: Calculate how many readings were recorded in the state of Arizona

Use your Python skills to calculate the number of readings that were recorded in the state of Arizona.

Expected output:

```
[OUT] 72
```

[5]:
```
### YOUR CODE HERE ###
len(aqi_dict['Arizona'])
```

[5]:
```
72
```

Hint

Because you only need a count of records, try using the `len()` function to check how many records are in the list that represents the value associated with the `'Arizona'` key.

1.2 2b: Calculate the mean AQI from the state of California

Use your Python skills to calculate the mean of the AQI readings that were recorded in the state of California. Note that there are many different approaches you can take. Be creative!

Expected output:

```
[OUT] 9.412280701754385
```

[6]:
```
### YOUR CODE HERE ###
ca_aqi_list = [aqi for county, aqi in aqi_dict['California']]
ca_aqi_mean = sum(ca_aqi_list) / len(ca_aqi_list)
ca_aqi_mean
```

[6]: 9.412280701754385

Hint 1

When you look up a state as a key in `aqi_dict`, its corresponding value is a list of tuples. Consider using a list comprehension to unpack the data contained in each tuple so you can perform operations on it.

Hint2

If you can create a list of just the AQI values from each tuple in the values of the 'California' key, then you can calculate the mean of the list in the next step.

Hint 3

One way to construct your list is:

`[aqi for county, aqi in aqi_dict['California']]`

Then, you can sum all the elements in the list and divide by the length of the list to calculate the mean.

2 Task 3: Define a `county_counter()` function

You want to be able to quickly look up how many times a county is represented in a given state's readings. Even though you already have a list containing just county names, it's not safe to rely on the counts from that list alone because some states might have counties with the same name. Therefore, you'll need to use the state-specific information in `aqi_dict` to calculate this information.

2.1 3a: Write the function

- Define a function called `county_counter` that takes one argument:
  - `state` - a string of the name of a U.S. state
- Return `county_dict` - a `dictionary` object whose keys are counties of the `state` given in the function's argument. For each county key, the corresponding value should be the count of the number of times that county is represented in the AQI data for that state.

Example:

```
[IN]  county_counter('Florida')
[OUT] {'Duval': 13,
       'Hillsborough': 9,
       'Broward': 18,
```

```
'Miami-Dade': 15,
'Orange': 6,
'Palm Beach': 5,
'Pinellas': 6,
'Sarasota': 9}
```

NOTE: Depending on the version of Python you're using, the order of the items returned by a dictionary can vary, so it's possible that your keys might not print in the same order as listed above. However, the key-value pairs themselves will be the same if you do the exercise successfully.

[7]:
```
### YOUR CODE HERE ###
def county_counter(state):
    county_dict = {}
    for county, aqi in aqi_dict[state]:
        if county in county_dict:
            county_dict[county] +=1
        else:
            county_dict[county] = 1
    return county_dict
```

Hint 1

Refer to what you learned about function syntax, how dictionaries behave with iteration techniques, and how to unpack tuples.

Hint 2

- Use `aqi_dict` in the body of the function.
- When you look up a state as a key in `aqi_dict`, its corresponding value is a list of tuples. Unpack the tuples to extract the county, which you can then use to build `county_dict`.

Hint 3

- Begin by instantiating `county_dict` as an empty dictionary.
- Then, in a `for` loop, unpack `county, aqi` from each tuple in `aqi_dict[state]`.
- Next, use conditional logic to assemble `county_dict`:
  - If `county` is already a key in `county_dict`, increment the value at that key by 1.
  - If `county` is not a key in `county_dict`, then assign it as a new key whose value is 1.
- Return `county_dict`.

2.2 3b: Use the function to check Washington County, PA.

Use the `county_counter()` function to calculate how many AQI readings were from `Washington` County, `Pennsylvania`.

Expected result:

```
[OUT] 7
```

[8]:
```
### YOUR CODE HERE ###
pa_dict = county_counter('Pennsylvania')
pa_dict['Washington']
```

[8]: 7

Hint

Use the `county_counter()` function to generate a dictionary of county counts for the state of Pennsylvania. Then, use the resulting dictionary to look up the count for Washington county.

2.3 3c: Use the function to check the different counties in Indiana

Use the `county_counter` function to obtain a list of all the different counties in the state of Indiana.

Expected result:

```
[OUT] dict_keys(['Marion', 'St. Joseph', 'Vanderburgh', 'Allen', 'Vigo'
      'Hendricks', 'Lake'])
```

NOTE: Depending on the version of Python you're using, the order of the items returned by a dictionary can vary, so it's possible that your keys might not print in this same order as listed above. However, the key-value pairs themselves will be the same if you do the exercise successfully.

[9]:
```
### YOUR CODE HERE ###
county_counter('Indiana').keys()
```

[9]:
```
dict_keys(['Marion', 'St. Joseph', 'Vanderburgh', 'Allen', 'Vigo', 'Hendricks',
'Lake'])
```

Hint 1

Refer to what you learned about dictionary methods.

Hint 2

Enter `'Indiana'` as an argument to your `county_counter()` function to get a dictionary of county counts in the state of Indiana. Then, use a dictionary method on the resulting dictionary.

Hint 3

Use the `keys()` dictionary method to return a list of the dictionary's keys, which represents the different counties in the state of Indiana.

3 Task 4: Use sets to determine how many counties share names

In this task, you'll create a list of every county from every state, then use it to determine how many counties have the same name.

3.1 4a: Construct a list of every county from every state

1. - Use `aqi_dict` and `county_counter()` to construct a list of every county from every state.
   - Assign the result to a variable called `all_counties`.
2. Find the length of `all_counties`.

Expected result:

```
[OUT] 277
```

Hint 1

Refer to what you learned about looping over dictionaries. Also, review previous tasks you completed in this notebook.

Hint 2

- Note that when you call the `keys()` method on a dictionary, the returned object is similar to a list, but it's not of type `list`. It's a dictionary view object, but you can convert it to a list using the `list()` function.
- Be mindful of the following:
  - `list_a.append(list_b)` increases the length of `list_a` by 1 no matter how long `list_b` is, because `append()` creates a list of lists.
  - `list_a + list_b` increases the length of `list_a` by `len(list_b)`, because adding (`+`) lists combines them.

Hint 3

1. Instantiate an empty list called `all_counties`.
2. Loop over each state in `aqi_dict.keys()`. For each iteration:
   - Extract a list of that state's counties using `list(county_counter(state).keys())`.
   - Add the list of counties to `all_counties`.
3. Print `len(all_counties)`

[10]:
```
# 1. ### YOUR CODE HERE ###
all_counties = []
for state in aqi_dict.keys():
    counties = list(county_counter(state).keys())
    all_counties += counties

# 2. ### YOUR CODE HERE ###
len(all_counties)
```

[10]:
```
277
```

3.2 4b: Calculate how many counties share names

Use `all_counties` and your knowledge of sets and list methods to determine how many counties share names.

Expected result:

```
[OUT] 41
```

[11]:
```
shared_count = 0

for county in set(all_counties):
    count = all_counties.count(county)
    if count > 1:
        shared_count += count

shared_count
```

[11]:
```
41
```

Hint 1

Refer to what you've learned about sets and review the list methods provided in Reference guide: Lists.

Hint 2

- A set cannot contain any duplicate elements. Each value is unique.
- Which list method returns the number of times an element occurs in a list?

Hint 3

1. Instantiate a counter with an initial value of `0` to record the number of counties with shared names.
2. Loop over each unique county name in `all_counties`. (Use the `set()` function to get unique names.)
3. For each iteration of the loop, use the `count()` list method to determine how many counties in `all_counties` have that county name.
4. If the number of counties with that name is more than one, add that number to the counter (from step 1).

Note that this doesn't tell you how many *different* county names were duplicated. Further analysis could uncover more details about this. Perhaps you can figure it out!

4 Conclusion

What are your key takeaways from this lab?

- Python has many built-in functions that are useful for building dictionaries and sets.
- Dictionaries in Python are useful for representing data in terms of keys mapped to values.

- A set will not allow duplicate values.
  - The values a set contains are unchangable and unordered.
- Functions and loop iteration can be used to perform calculations on dictionary values.
  - Once the values have been calculated, they can be saved to other data types, such as tuples, lists, and sets.
- There are many ways to access data stored inside a dictionary.

Congratulations! You've completed this lab. However, you may not notice a green check mark next to this item on Coursera's platform. Please continue your progress regardless of the check mark. Just click on the "save" icon at the top of this notebook to ensure your work has been logged.