Course Project - Milestone 1

Processing Data from Real-World Data Sets

Topics Covered: Lists, Dictionaries, Loops, Conditionals (Lectures up to Day 21 on Oct 18)

Overview

In this project you will be shown how to incrementally build a small application to visualize real world data in Python. By the end of the project your application will be able to pull live data from the web, clean and filter that data, cache it locally, and visualize it. The project will use a dataset the instructor selected from among the datasets published on the Buffalo's OpenData website.

Later parts of the project will have you writing code that downloads data from the OpenData website, stores local copies of that data, and visualizes the data. As with all large projects, breaking up the project into smaller pieces makes writing and testing each piece easier and completing the project more manageable. For this first step, you will write the functions which process a dataset to generate the numbers we will need. The final project will use real data and so "correct" results will have to change with the release of each day's data. These daily updates are critical for the professionals relying on these data, but complicate the steps needed to check that our code works properly. Following common practice, you can use our sample input (found as an assignment statement in the Sample Data For Testing section below) to begin testing your code. This code declares a variable named **`data`** and assigns to it a list of dictionaries.

Remember that the majority of the project points are earned by your final project submission. So it is important to complete these functions, even if you miss the Part 1 submission deadline.

This is an INDIVIDUAL project. You cannot work with other people on this project.

Submission and Grading

The submission for this Milestone is due Friday, Nov. 3 @ 11:59PM
This submission is worth 5 out of the project's 60 total points
You must submit to **AutoLab** to receive credit

AutoLab will limit the number of times you submit to each of the submission targets; you will be able to make at most 5 (five) submissions to each.

Functions to be Graded

Described below are the five functions that AutoLab will evaluate for Part 1. While you are not required to write any additional functions, we are happy for you to do so if it helps you complete this work.

`getValuesForKey`

Define a function named **`getValuesForKey`** with two parameters:

- the 1st parameter, called **`key`** in this description, is a string (a key);
- the 2nd parameter, called **`records`** in this description, is a list of dictionaries (the data);

It should return a list of all values stored in **`records`** associated with the key **`key`**.

It should use the accumulator pattern to create and return a new list. The accumulator loop should iterate over the dictionaries in **`records`**. For the accumulator pattern's update step you should first get the value that corresponds to the **`key`**. You should then check if that value is not already in the accumulator list and, when it is not in the list, add it.

You CAN NOT assume that all dictionaries in the second parameter will contain the passed in key.

Sample function call:
`getValuesForKey('Artist', data)` should return a list of all of the different artists in **`data`**.

`countMatchesByKey`

Define a function named **`countMatchesByKey`** with three parameters:

- the 1st parameter, called **`key`** in this description, is a string (a key);
- the 2nd parameter, called **`value`** in this description, is a value (a value);
- the 3rd parameter, called **`records`** in this description, is a list of dictionaries (the data);

It should return the number of records whose value associated with the passed in **`key`** matches the passed in **`value`**.

It should use the accumulator pattern to calculate and return an int. It should initialize the accumulator variable to 0. The accumulator loop should iterate over the dictionaries in **`records`**. Inside the loop, if the current dictionary's value associated with the key, **`key`**, is equivalent to **`value`**, increase the accumulator variable's value by one.

You CAN NOT assume that all dictionaries in the third parameter will contain the passed in key.

Sample function call:
`countMatchesByKey('Artist', 'BIFF HENRICH', data)` should return the number of pieces by Biff Henrich in **`data`**.

`countMatchesByKeys`

Define a function named **`countMatchesByKeys`** with five parameters:

- the 1st parameter, called **`key1`** in this description, is a string (a key);
- the 2nd parameter, called **`value1`** in this description, is a value (a value);
- the 3rd parameter, called **`key2`** in this description, is a string (a key);
- the 4th parameter, called **`value2`** in this description, is a value (a value);
- the 5th parameter, called **`records`** in this description, is a list of dictionaries (the data);

It should return the number of records whose value associated with the **`key1`** matches **`value1`** AND whose value associated with **`key2`** matches the **`value2`** .

It should use the accumulator pattern to calculate and return an int. It should initialize the accumulator variable to 0. The accumulator loop should iterate over the dictionaries in **`records`**. Inside the loop, if the current dictionary's value associated with the **`key1`** is equivalent to **`value1`** AND the value associated with the **`key2`** is equivalent to **`value2`**, increase the accumulator variable's value by one.

You CAN NOT assume that all dictionaries in the fifth parameter will contain either of the passed in keys.

Sample function call:
`countMatchesByKeys('Artist', 'JOSHUA G STEIN', 'Category', 'SCULPTURE', data)` should return the number of sculptures by Joshua G Stein in **`data`**.

`filterByKey`

Define a function named **`filterByKey`** with three parameters:

- the 1st parameter, called **`key`** in this description, is a string (a key);
- the 2nd parameter, called **`value`** in this description, is a value (a value);
- the 3rd parameter, called **`records`** in this description, is a list of dictionaries (the data);

It should return a new list of dictionaries corresponding to the records whose value associated with the passed in **`key`** matches the passed in **`value`**.

It should use the accumulator pattern to create and return a list of dictionaries. It should initialize the accumulator variable to an empty list. The accumulator loop should iterate over the dictionaries in **`records`**. Inside the loop, if the current dictionary's value associated with the key, **`key`**, is equivalent to **`value`**, it should append the dictionary to the accumulator variable.

You CAN NOT assume that all dictionaries in the third parameter will contain the passed in key.

Sample function call:
`filterByKey('Category', 'SCULPTURE', data)` should return a list of all the sculptures in **`data`**.

`computeFrequency`

Define a function named **`computeFrequency`** with two parameters:

- the 1st parameter, called **`key`** in this description, is a string (a key);
- the 2nd parameter, called **`records`** in this description, is a list of dictionaries (the data);

It should return a dictionary where the keys are all of the values in **`records`** associated with **`key`**, and the values are the number of times that value appears in **`records`**.

It should use the accumulator pattern to create and return a new dictionary. The initial value for your accumulator variable should be an empty dictionary (e.g., {}). For the accumulator pattern's update step, get the value associated with **`key`**. If that value is not a key in your accumulator variable, then add it as a key to your accumulator with a value of 0. If it is already a key, then update the associated value by adding one.

You CAN NOT assume that all dictionaries in the second parameter will contain the passed int key. You should ONLY update the accumulator when **`key`** is a key in a dictionary.

Sample function call:
`computeFrequency ('Category', data)` should return a dictionary where the keys are all of the different art categories, and the value are the number of times that category shows up in data.

For example if there are 10 sculptures and 24 paintings in data, the return value would be:
`{'SCULPTURE' : 10, 'PAINTING' : 24}`

Sample Data For Testing

```
data = [ {'Title': 'SOLE PARK', 'Category': 'SCULPTURE', 'Type': 'RELIEF',
'Medium': 'STONE', 'Frame': 'false', 'Photo URL Link': 'UNKNOWN', 'Artist':
'UNKNOWN', 'Street Address': 'BUSTI AVE & NIAGARA ST', 'City': 'BUFFALO',
'Zipcode': '14213', 'State': 'NY'},
{'Title': 'BUFFALO STREET MAP', 'Category': 'GRAPHIC ARTS', 'Type': 'MAP',
'Medium': 'PARCHMENT', 'Frame': 'true', 'Photo URL Link': 'UNKNOWN',
'Artist': 'SMITH BROTHERS COMPANY', 'Street Address': '65 NIAGARA SQUARE',
'City': 'BUFFALO', 'Zipcode': '14202', 'State': 'NY'},
{'Title': 'WAR  MEMORIAL STADIUM RENDERING', 'Category': 'GRAPHIC ARTS',
'Type': 'DRAWING', 'Medium': 'PAPER', 'Frame': 'false', 'Photo URL Link':
'UNKNOWN', 'Artist': 'UNKNOWN', 'Street Address': '65 NIAGARA SQUARE',
'City': 'BUFFALO', 'Zipcode': '14202', 'State': 'NY'},
{'Title': 'MAYOR HIRAM BARTON', 'Category': 'PAINTINGS', 'Type': 'PORTRAIT',
'Medium': 'OIL ON CANVAS', 'Frame': 'true', 'Photo URL Link':
'HTTP://WWW.CI.BUFFALO.NY.US/FILES/1_2_1/PUBLIC%20ART%20WEBSITE/WEB%20PAGES/H
IRAM%20BARTON.HTML', 'Artist': 'UNKNOWN', 'Street Address': '65 NIAGARA
SQUARE', 'City': 'BUFFALO', 'Zipcode': '14202', 'State': 'NY'},
{'Title': 'MAYOR ELI COOK', 'Category': 'GRAPHICS ARTS', 'Type': 'PORTRAIT',
'Medium': 'PASTEL ON PAPER', 'Frame': 'true', 'Photo URL Link':
'HTTP://WWW.CI.BUFFALO.NY.US/FILES/1_2_1/PUBLIC%20ART%20WEBSITE/WEB%20PAGES/E
LI%20COOK.HTML', 'Artist': 'UNKNOWN', 'Street Address': '65 NIAGARA SQUARE',
'City': 'BUFFALO', 'Zipcode': '14202', 'State': 'NY'},
{'Title': 'MAYOR ANTHONY MASIELLO', 'Category': 'PAINTINGS', 'Type':
'PORTRAIT', 'Medium': 'OIL ON CANVAS', 'Frame': 'true', 'Photo URL Link':
'UNKNOWN', 'Artist': 'NATHAN NAETZKER', 'Street Address': '65 NIAGARA
SQUARE', 'City': 'BUFFALO', 'Zipcode': '14202', 'State': 'NY'},
{'Title': 'MAYOR CHANDLER J WELLS', 'Category': 'PAINTINGS', 'Type':
'PORTRAIT', 'Medium': 'OIL ON CANVAS', 'Frame': 'true', 'Photo URL Link':
'HTTP://WWW.CI.BUFFALO.NY.US/FILES/1_2_1/PUBLIC%20ART%20WEBSITE/WEB%20PAGES/C
HANDLER%20WELLS.HTML', 'Artist': 'ALVAH BRADISH', 'Street Address': '65
NIAGARA SQUARE', 'City': 'BUFFALO', 'Zipcode': '14202', 'State': 'NY'} ]
```