Lab #4

Due: 11/7/22 @ 11:59pm

Content Covered

Reading and Writing CSV files in Python, Lists and Dictionaries in Python

Overview

Labs 4 and 5 will be a bit different than the previous 3. In labs 4 and 5 you will build a complete web application which can serve as a model for how to structure your project.

Each lab guides you through the process of writing a small web application. The web application produces a visualization in a browser front end, of data provided by a back end web server. Together they bring together all the elements of the course we've covered so far, including but not limited to reading and writing files, representing data using data structures, processing data (e.g. filtering, combining), using libraries, and setting up a simple web server.

Successful completion requires that a number of small components be assembled in a way that they can all work together. I do not expect you will be able to complete this without some guidance. **Do not be shy to take advantage of the Piazza forum and office hours to discuss issues you run into.**

Lab 4 will handle the application code (the code that actually does the work: reading/writing files, manipulating data, etc). Lab 5 will handle the web server backend, front end visualization, and integration of all the pieces into one coherent WebApp.

Getting Started

1. Sign into replit.com
2. Create a new Python REPL. **Unlike previous Labs, all of the functions you write for this Lab must be defined in a file named `App.py`.**
    a. Since repl.it runs the code in `main.py`, if you want to test what you are working on for this lab, you can put your testing code in `main.py`, and import your application code. There is a Piazza post related to this process here.
3. In this lab we will process data from the 311 Service Requests data set available via OpenData Buffalo. This data set has over 800,000 data records. For ease of testing, you can work with an abbreviated version of this data set with a little over 1,000 data records available here.
    a. During your testing, it may be useful to create even smaller versions of the file to use. You can do so by editing the file in any spreadsheet application to get a feel for the data, and modify/shrink down the file.

Editing Your Code

In this lab you will be defining a number of functions to perform various tasks in your web application. All functions should be defined in `App.py`. It is important that you name your functions exactly as specified by each problem so that the autograder will be able to call them.

Any variables and parameters can be named however you like. You can write any additional functions to help you solve the problems as you see fit. Each problem is more complex than the ones before it.

Submission

The Autolab submission for this lab will open up no later than 10/31/22 @ 5:00pm.

Your final submission for this lab will be a .zip file downloaded from replit.com and submitted to Autolab. You may submit as many times as you like, but only your last submission will count for your final grade.

Submissions close at 11:59pm on 11/7/22 and no late submissions will be accepted.

Functions

---

readCSV

Define the function `readCSV(`*filename*`)` so it reads the contents of the CSV file named *filename* and returns a list of dictionaries corresponding to its contents. The CSV file will have a header file with field names. Each dictionary in the returned list of dictionaries must consist of key-value pairs where the key is a field name. The field name must be paired with the corresponding data value from the record.

For example, if `sample.csv` contains:

```
a,b,c
1,2,3
4,5,6
7,8,9
```

then `readCSV('sample.csv')` must return this list:

```
[ { 'a':'1', 'b':'2', 'c':'3' },
  { 'a':'4', 'b':'5', 'c':'6' },
  { 'a':'7', 'b':'8', 'c':'9' }
]
```

NOTE: A pedagogical goal of this exercise is to give you additional practice reading from a CSV file, writing loops, working with lists and dictionaries, and writing accumulation code. Therefore you are NOT permitted to simply use the csv.DictReader to complete this task.

NOTE: The testing file used on AutoLab has UTF-8 encoded characters. To correctly read them you must open the file with encoding='utf-8' in the open command:

```
open(filename, newline='', encoding="utf-8")
```

---

writeCSV

Define the function `writeCSV`(*filename, list_of_dictionaries*) so it writes data contained in *list_of_dictionaries* as a CSV file named *filename* and returns `None`. If the file already exists it must be overwritten with its new content. If the file does not already exist it must be created with the new content. *list_of_dictionaries* will contain a list of dictionaries in the format returned by `readCSV`. The CSV file written to *filename* must have a header row containing the field names, followed by all the data rows represented by the values in each dictionary.

For example, if `lod` is:

```
[ { 'abby':'51', 'bell':'62', 'dee':'33', 'river':'97' },
  { 'abby':'54', 'bell':'65', 'dee':'26', 'river':'71' },
  { 'abby':'45', 'bell':'39', 'dee':'88', 'river':'22' },
  { 'abby':'57', 'bell':'68', 'dee':'39', 'river':'62' }
]
```

then `writeCSV('output.csv', lod)` must ensure that a file name *output.csv* has the following contents:

```
abby,bell,dee,river
51,62,33,97
54,65,26,71
45,39,88,22
57,68,39,62
```

NOTE: A pedagogical goal of this exercise is to give you additional practice writing loops, working with lists and dictionaries, and writing to a file. Therefore you are NOT permitted to simply use the csv.DictWriter to complete this task.

keepOnly

Define the function `keepOnly(`*list_of_dictionaries*`,` *key*`,` *value*`)` so it returns a list of dictionaries which contains only those dictionaries from *list_of_dictionaries* which contain the *key:value* pairing.

For example, if `lod` is:

```
[ { 'abby':'51', 'bell':'62', 'dee':'33', 'river':'97' },
  { 'abby':'54', 'bell':'65', 'dee':'26', 'river':'71' },
  { 'abby':'45', 'bell':'39', 'dee':'88', 'river':'22' },
  { 'abby':'57', 'bell':'68', 'dee':'26', 'river':'62' }
]
```

then `keepOnly(lod, 'dee', '26')` must return

```
[ { 'abby':'54', 'bell':'65', 'dee':'26', 'river':'71' },
  { 'abby':'57', 'bell':'68', 'dee':'26', 'river':'62' }
]
```

---

discardOnly

Define the function `discardOnly(`*list_of_dictionaries*`,` *key*`,` *value*`)` so it returns a list of dictionaries which contains only those dictionaries from *list_of_dictionaries* which do not contain the *key:value* pairing.

For example, if `lod` is:

```
[ { 'abby':'51', 'bell':'62', 'dee':'33', 'river':'97' },
  { 'abby':'54', 'bell':'65', 'dee':'26', 'river':'71' },
  { 'abby':'45', 'bell':'39', 'dee':'88', 'river':'22' },
  { 'abby':'57', 'bell':'68', 'dee':'26', 'river':'62' }
]
```

then `discardOnly(lod, 'dee', '26')` must return

```
[ { 'abby':'51', 'bell':'62', 'dee':'33', 'river':'97' },
  { 'abby':'45', 'bell':'39', 'dee':'88', 'river':'22' }
]
```

---

filterRange

Define the function `filterRange(`*list_of_dictionaries*`,` *key*`,` *low*`,` *high*`)` so it returns a list of dictionaries which contains only those dictionaries *d* from *list_of_dictionaries* for which *low* <= *d[key]* < *high*

For example, if `lod` is:

```
[ { 'abby':'51', 'bell':'62', 'dee':'33', 'river':'97' },
  { 'abby':'54', 'bell':'65', 'dee':'26', 'river':'71' },
  { 'abby':'45', 'bell':'39', 'dee':'88', 'river':'22' },
  { 'abby':'57', 'bell':'68', 'dee':'26', 'river':'62' }
]
```

then `filterRange(lod, 'bell', '39', '68')` must return

```
[ { 'abby':'51', 'bell':'62', 'dee':'33', 'river':'97' },
  { 'abby':'54', 'bell':'65', 'dee':'26', 'river':'71' },
  { 'abby':'45', 'bell':'39', 'dee':'88', 'river':'22' }
]
```

---

duration

The datetime Python module makes it easy to do computations with dates. For this project you will need to compute the duration of a request (each record in the CSV file, and therefore each dictionary in the list of dictionaries, corresponds to one request).

You can read the documentation for the datetime module, but what you need to know is how to create a date object, how to compute the difference between two dates (called a timedelta), and how to get the number of days in a timedelta. This short example should give you the basic idea:

```
> import datetime
> d1 = datetime.date(2016,5,3)
> d2 = datetime.date(2016,11,21)
> diff = d2 - d1
> diff
datetime.timedelta(days=202)
> diff.days
202
```

Define the function `duration(`*date1*`,` *date2*`)` so that it accepts two datetime.date objects as arguments and return the number of days between the two dates, as an integer.

Useful Suggestions and Advice

- **Think before you code** - think about the input to the function, and the desired output, and plan out how you would solve the problem in your head or on paper before you start to code. Having an idea of where you are going before you start typing can make the coding process smoother.
- **Write your own tests** - each problem describes an example test case or two that you can use to check if your implementation is correct. But they are not sufficient to guarantee you have a completely correct implementation. It can help to think through and write additional test cases (even before you code) to make sure your functions are behaving as you expect them to. This will also allow you to more quickly test your code as you go, rather than having to wait for Autolab to be open, and to go through the entire submission process just to get feedback.
- **Break down bigger problems into smaller ones** - if you are stuck on a problem, try solving a smaller piece of the problem first. You can even write it as a separate function, and test it out. Then once you have the small pieces working, building the solution to the bigger problem from your smaller solutions should be easier.
- **Ask for help** - during lab sessions, in office hours, and on Piazza. If you do choose to ask questions on Piazza **do not post your own code in public posts.** If you are going to make your question public, make sure it is general/conceptual in nature. Otherwise, make your question private to just the instructors.