Assignment01

September 10, 2018

1 CS 594 / CS 690 - Assignment 01

1.1 ### August 27, 2018

For this assignment, you must work in groups of one or two students. Each person is responsible to write their own code, but the group will (together) discuss their solution. In this notebook, we provide you with basic functions for completing the assignment. *You will need to modify existing code and write new code to find a solution*. Each member of the group must upload their own work to GitHub (which we will cover in the next lecture).

2 Problem 1

In this problem we will explore reading in and parsing delimiter-separated values stored in files. We will start with comma-separated values and then move on to tab-separated values.

2.0.1 Problem 1a: Comma-Separated Values (CSV)

From Wikipedia: In computing, a comma-separated values (CSV) file stores tabular data (numbers and text) in plain text. Each line of the file is a data record. Each record consists of one or more fields, separated by commas. The use of the comma as a field separator is the source of the name for this file format.

If you were to consider the CSV file as a matrix, each line would represent a row and each comma would represent a column. In the provided CSV file, the first row consists of a header that "names" each column. In this problem, ...

- Count (and print) the number of rows of data (header is excluded) in the csv file
- Count (and print) the number of columns of data in the csv file
- Calculate (and print) the average of the values that are in the "age" column
- You can assume each age in the file is an integer, but the average should be calculated as a float

```
In [37]: def parse_delimited_file(filename, delimiter=","):
             # Open and read in all lines of the file
             # (I do not recommend readlines for LARGE files)
             # `open`: ref [1]
             # `readlines`: ref [2]
             with open(filename, 'r', encoding='utf8') as dsvfile:
```

```
        lines = dsvfile.readlines()

    # Strip off the newline from the end of each line
    # HINT: ref [3]
    # Using list comprehension is the recommended pythonic way to iterate through lis
    # HINT: ref [4]
    new_list = []
    for line in lines:
        new_list.append(line.rstrip())

    # Split each line based on the delimiter (which, in this case, is the comma)
    # HINT: ref [5]


    # Separate the header from the data
    # HINT: ref [6]
    first_row = new_list.pop(0)

    # Find "age" within the header
    # (i.e., calculating the column index for "age")
    # HINT: ref [7]
    headers = first_row.split(delimiter)
    age_index = headers.index('age',0)


    # Calculate the number of data rows and columns
    # HINT: [8]
    num_data_rows = len(new_list)
    num_data_cols = len(headers)

    # Sum the "age" values
    # HINT: ref [9]
    age_sum = 0
    for data in new_list:
        columns = data.split(delimiter)
        age = columns[age_index].lstrip()
        age_sum = age_sum + int(age)

    # Calculate the average age
    ave_age = age_sum / num_data_rows

    # Print the results
    # `format`: ref [10]
    print("Number of rows of data: {}".format(num_data_rows))
    print("Number of cols: {}".format(num_data_cols))
    print("Average Age: {}".format(ave_age))

# Parse the provided csv file
```

```
parse_delimited_file('data.csv')
```

```
Number of rows of data: 8
Number of cols: 3
Average Age: 70.875
```

Expected Ouput:

```
Number of rows of data: 8
Number of cols: 3
Average Age: 70.875
```

References: - 1: open - 2: readlines - 3: rstrip - 4: list comprehension - 5: split - 6: splice - 7: "more on lists" - 8: len - 9: int - 10: format

2.0.2 Problem 1b: Tab-Separated Values (TSV)

From Wikipedia: A tab-separated values (TSV) file is a simple text format for storing data in a tabular structure, e.g., database table or spreadsheet data, and a way of exchanging information between databases. Each record in the table is one line of the text file. Each field value of a record is separated from the next by a tab character. The TSV format is thus a type of the more general delimiter-separated values format.

In this problem, repeat the analyses performed in the prevous problem, but for the provided tab-delimited file.

NOTE: the order of the columns has changed in this file. If you hardcoded the position of the "age" column, think about how you can generalize the `parse_delimited_file` function to work for any delimited file with an "age" column.

```
In [36]: def parse_delimited_file(filename, delimiter=","):
             # Open and read in all lines of the file
             # (I do not recommend readlines for LARGE files)
             # `open`: ref [1]
             # `readlines`: ref [2]
             with open(filename, 'r', encoding='utf8') as dsvfile:
                 lines = dsvfile.readlines()

             # Strip off the newline from the end of each line
             # HINT: ref [3]
             # Using list comprehension is the recommended pythonic way to iterate through lis
             # HINT: ref [4]
             new_list = []
             for line in lines:
                 new_list.append(line.rstrip())

             # Split each line based on the delimiter (which, in this case, is the comma)
             # HINT: ref [5]
```

```
        # Separate the header from the data
        # HINT: ref [6]
        first_row = new_list.pop(0)

        # Find "age" within the header
        # (i.e., calculating the column index for "age")
        # HINT: ref [7]
        headers = first_row.split(delimiter)
        age_index = headers.index('age',0)


        # Calculate the number of data rows and columns
        # HINT: [8]
        num_data_rows = len(new_list)
        num_data_cols = len(headers)

        # Sum the "age" values
        # HINT: ref [9]
        age_sum = 0
        for data in new_list:
            columns = data.split(delimiter)
            age = columns[age_index].lstrip()
            age_sum = age_sum + int(age)

        # Calculate the average age
        ave_age = age_sum / num_data_rows

        # Print the results
        # `format`: ref [10]
        print("Number of rows of data: {}".format(num_data_rows))
        print("Number of cols: {}".format(num_data_cols))
        print("Average Age: {}".format(ave_age))


    # Further reading on optional arguments, like "delimiter": http://www.diveintopython.
    parse_delimited_file('data.tsv', delimiter="\t")
```

```
Number of rows of data: 8
Number of cols: 3
Average Age: 70.875
```

Expected Ouput:

```
Number of rows of data: 8
Number of cols: 3
Average Age: 70.875
```

---

3 Problem 2

If you opened the `data.csv` file, you may have noticed some non-english letters in the names column. These characters are represented using Unicode, a standard for representing many different types and forms of text. Python 3 natively supports Unicode, but many tools do not. Some tools require text to be formatted with ASCII.

Convert the unicode-formatted names into ascii-formated names, and save the names out to a file named `data-ascii.txt` (one name per line). We have provided you with a tranliteration dictionary that maps several common unicode characters to their ascii transliteration. Use this dictionary to convert the unicode strings to ascii.

```
In [60]: translit_dict = {
             "ä" : "ae",
             "ö" : "oe",
             "ü" : "ue",
             "Ä" : "Ae",
             "Ö" : "Oe",
             "Ü" : "Ue",
             "" : "l",
             "" : "o",
         }

         with open("data.csv", 'r', encoding='utf8') as csvfile:
             lines = csvfile.readlines()

         # Strip off the newline from the end of each line
         new_list = []
         for line in lines:
             new_list.append(line.rstrip())

         # Split each line based on the delimiter (which, in this case, is the comma)



         # Separate the header from the data
         first_row = new_list.pop(0)

         # Find "name" within the header
         headers = first_row.split(',')
         name_index = headers.index('name',0)

         # Extract the names from the rows
         unicode_names = []
         for data in new_list:
             columns = data.split(',')
             unicode_names.append(columns[name_index])
```

```
        # Iterate over the names
        translit_names = []
        for unicode_name in unicode_names:
            # Perform the replacements in the translit_dict
            # HINT: ref [1]
            ascii_name = unicode_name
            for key in translit_dict:
                ascii_name = ascii_name.replace(key, translit_dict[key])
            translit_names.append(ascii_name)

        # Write out the names to a file named "data-ascii.txt"
        # HINT: ref [2]
        f = open("data-ascii.txt","w+")
        for name in translit_names:
            f.write(name + "\n")

        # Verify that the names were converted and written out correctly
        with open("data-ascii.txt", 'r') as infile:
            for line in infile:
                print(line.rstrip())
```

```
Richard Phillips Feynman
Shin'ichiro Tomonaga
Julian Schwinger
Rudolf Ludwig Moessbauer
Erwin Schroedinger
Paul Dirac
Maria Sklodowska-Curie
Pierre Curie
```

Expected Output:

```
Richard Phillips Feynman
Shin'ichiro Tomonaga
Julian Schwinger
Rudolf Ludwig Moessbauer
Erwin Schroedinger
Paul Dirac
Maria Sklodowska-Curie
Pierre Curie
```

References: - 1: replace - 2: file object methods

4 Free-Form Questions:

Answer the following questions, in a couple sentences each, in the cells provided below.

- Your solutions for Problems 1 & 2 probably share a lot of code in common. You might even have copied-and-pasted from Problem 1 into Problem 2. How would you refactor `parse_delimited_file` to be useful in both problems?
- Are there any pre-built Python packages that could help you solve these problems? How could you use them?
- List the key tasks you accomplished during this assignment.
- Describe the challenges you faced in addressing these tasks and how you overcame these challenges.
- Did you work with other students on this assignment? If yes, how did you help them? How did they help you? Be as specific as possible.

1. I would write a new method for replacing the substring. Then, from the 'parse_delimited_file' method I would call the replacing method.

2. There is a module named csv. We can import the csv module and use the csv.reader() method to make our task easier.

3. - Learned some git commands

- Learned some new Syntax of Python
- Installed Anaconda in Ubuntu and Mac

4. The only challenge I faced to make my laptop ready for the classroom. Unfortunately, I couldn't still figure it out. I'm working in my Lab PC and Home PC. I think, I need to buy a new laptop but it will take time.

5. No, I worked myself.