# INDUSTRIAL PROGRAMING

Coursework 2

DECEMBER 6, 2015

MOHAMED SHARIF
H00158013

# Table of Contents

# Introduction

The assignment was to develop a simple data analysis application that can read and parse JSON files and then to apply certain analysis functions to it-. This report will show the applications features, design considerations, developers/user guide and testing the application in different conditions. The assumptions made during the development was to read the file line by line where each line represents a JSON object. The environment used during the development was PyCharm and the programming language used is Python 3.

# Requirements Checklist

1. **Python:** The core logic of the application should be implemented in Python. [Complete]
2. **Views by country/continent:** We want to analyse, for a given document, from which countries and continents the document has been viewed. The data should be displayed as a histogram of countries, i.e. counting the number of occurrences for each country in the input file.
    a. The application should take a string as input, which uniquely specifies a document (a document UUID), and return a histogram of countries of the viewers. The histogram can be displayed using mathplot. [Complete]
    b. Use the data you have collected in the previous task, group the countries by continent, and generate a histogram of the continents of the viewers. The histogram can be displayed using mathplot. [Complete]
3. **Views by browser:** In this task we want to identify the most popular browser. To this end, the application has to examine the visitor useragent field and count the number of occurrences for each value in the input file.
    a. The application should return and display a histogram of all browser identifiers of the viewers. [Complete]
    b. In the previous task, you will see that the browser strings are very verbose, distinguishing browser by e.g. version and OS used. Process the input of the above task, so that only the main browser name is used to distinguish them (e.g. Mozilla), and again display the result as a histogram. [Complete]
4. **Reader profiles:** In order to develop a readership profile for the site, we want to identify the most avid readers. We want to determine, for each user, the total time spent reading documents. The top 10 readers, based on this analysis, should be printed. [Complete]
5. **“Also likes” functionality:** Popular document-hosting web sites, such as Amazon, provide information about related documents based on document tracking information. One such feature is the “also likes” functionality: for a given document, identify, which other documents have been read by this document’s readers. The idea is that, without examining the detail of either document, the information that both documents have been read by the same reader relates two documents with each other. Figure 1 gives an example of

this functionality. In this task, you should write a function that generates such an “other readers of this document also like” list, which is parametrised over the function to determine the order in the list of documents. Display the top 10 documents, which are “liked” by other readers.

   a. Implement a function that takes a document UUID and returns all visitor UUIDs of readers of that document. [Complete]
   b. Implement a function that takes a visitor UUID and returns all document UUIDs that have been read by this visitor. [Complete]
   c. Using the two functions above, implement a function to implement the “also like” functionality, which takes as parameters the above document UUID and visitor UUID, and additionally a sorting function on documents. The function should return a list of “liked” documents, sorted by the sorting function parameter. Note: the implementation of this function must not fix the way how documents are sorted. [Complete]
   d. Use this function to produce an “also like” list of documents, using the above function, based on readership profile for sorting the documents. [Complete]
   e. Use this function to produce an “also like” list of documents, using a sorting function, based on the number of readers of the same document. [Complete]
   f. In each of the above two use cases, provide a document UUID and visitor UUID as input and produce a list of top 10 document UUIDs as a result. [Complete]

6. **GUI usage:** To read the required data and to display the statistical data, develop a simple GUI that reads the user inputs described above, and with buttons to process the data as required per task. [Complete]
7. **Command-line usage:** The application shall provide a command-line interface to test its functionality in an automated way, like this:

```
% cw2 -u user_uuid -d doc_uuid -t task_id
```

to check the results of implementing task task_id using inputs user_uuid for the user UUID and doc_uuid for the document UUID. [Complete]

# Design Considerations

## Class Design

Classes have an appropriate identifier for the elements that make up class shown below along with adequate commentary within the code. A class for each feature is implemented displayed bellow.

- GUI (gui.py)
- Reader (reader.py)
- Parsing JSON (Json_parser.py)
- Plotting Histograms (histograms.py)
- Task Management (task_manger.py)

## GUI Design

Consideration was taken while designing the Graphic user interface of the application. The aim of the GUI design was to create an interface that is easy to use and be readable.

The overall design used flat simple style to give it the feel of a modern application and to make it more appealing to use.

Navigation button were implemented shown in below was to allow the user to choose tasks they wanted to test and read new files if required. The buttons are an appropriate size to increase the accuracy of clicking.

Either the chart or list of results is displayed at one time in the interface to avoid mixing the results.

## Data Structures Used

Dictionaries were used to store data in most cases this was a better alternative to matrices as it helps improve performance due to using keys.

Lists were used to store some of the results in which dictionaries are deemed redundant. For example when we wanted to sort the top 10 users.

## Advanced Language Feature and Functionalities

High order functions were used to help improve coding style and to reduce the repetition of code.

Exceptions were used to catch error cases and handle them appropriately.

# User Guide

The application can be used with two different interfaces, command line interface and GUI interface. The application is launched using command line in both cases.

## Command Line Interface

The application can be launched using the following command

```
% cw2 -u user_uuid -d doc_uuid -t task_id –f filename
```

Where depending on the task the arguments are supplied/ignored

For tasks 3a, 3b and 4 this is all that is required to run the application

```
% cw2 -t task_id
```

For tasks 2a, 2b and 5a this is all that is required to run the application

```
% cw2 -d doc_uuid -t task_id
```

For task 5b this is all that is required to run the application

```
% cw2 -u user_uuid -t task_id
```

For tasks 5c, 5e and 5d this is all that is required to run the application

```
% cw2 -u user_uuid -d doc_uuid -t task_id
```

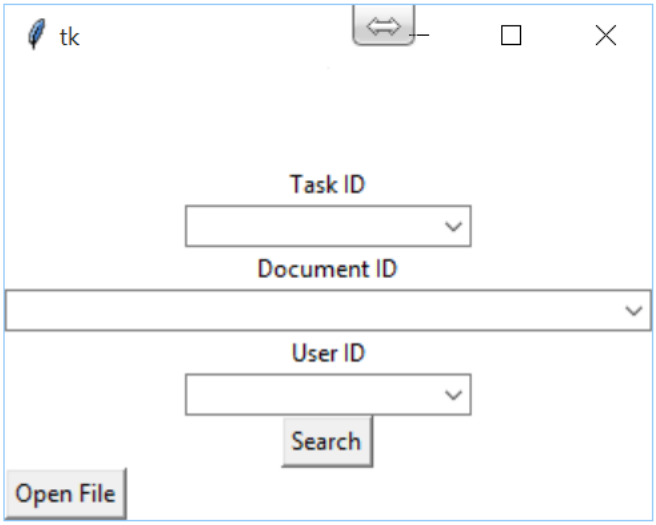
In all cases if a different file is required to be used just use the –f “filename” argument to supply the file.

## GUI Interface

To run the application in GUI mode just use the following command.

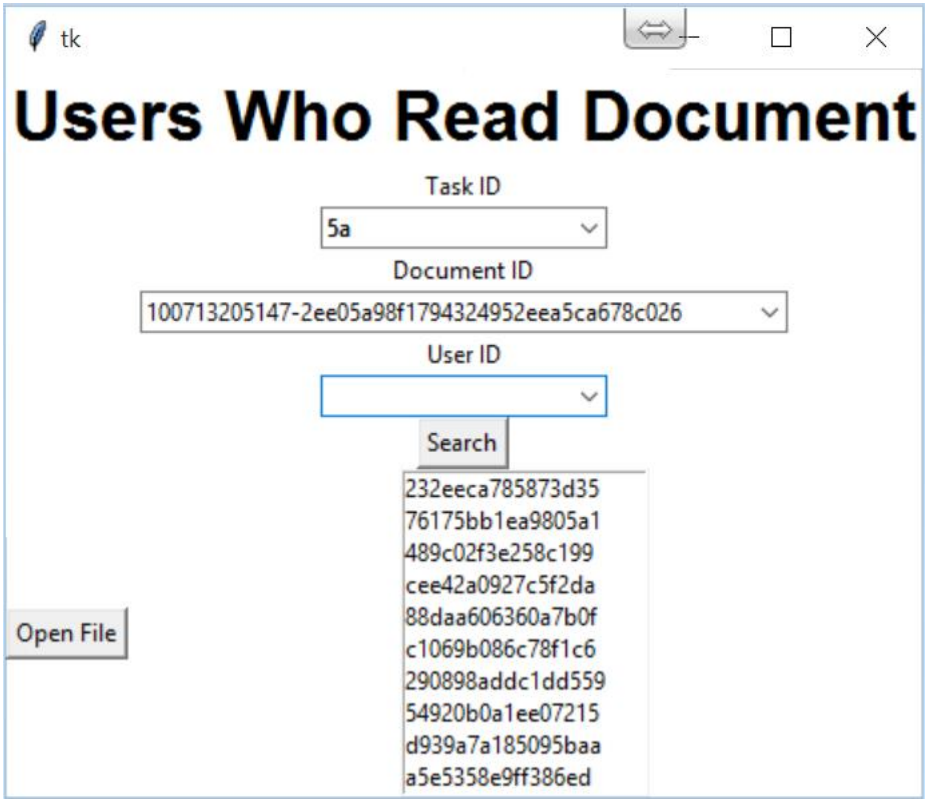
```
% cw2 –g yes
```

The following window will appear.

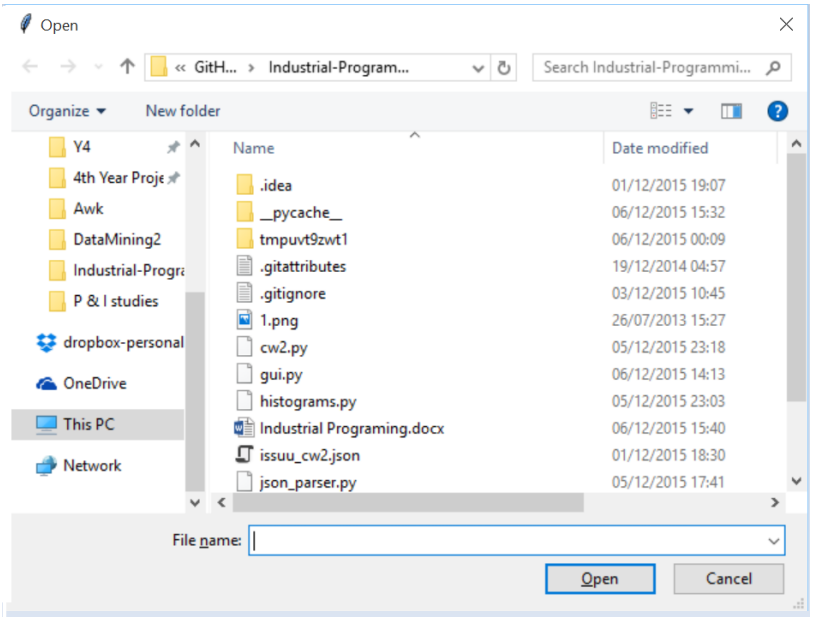



Choose a task from the task ID options menu, do the same for the rest of the menus based on which task some menus are required as explained in the previously.

Click the search button for the result of the query to show up this can take some time for some tasks, as they require a lot of data manipulation, below an example can be seen of task 5a.




Opening a file can be done by clicking on the open file button and choosing a file using the open file dialog pop up, as seen in the figure below.


Open
« GitH... › Industrial-Program...
Search Industrial-Programmi...
Organize
New folder
Y4
4th Year Proje
Awk
DataMining2
Industrial-Progra
P & I studies
dropbox-personal
OneDrive
This PC
Network
Name
Date modified
.idea 01/12/2015 19:07
__pycache__ 06/12/2015 15:32
tmpuvt9zwt1 06/12/2015 00:09
.gitattributes 19/12/2014 04:57
.gitignore 03/12/2015 10:45
1.png 26/07/2013 15:27
cw2.py 05/12/2015 23:18
gui.py 06/12/2015 14:13
histograms.py 05/12/2015 23:03
Industrial Programing.docx 06/12/2015 15:40
issuu_cw2.json 01/12/2015 18:30
json_parser.py 05/12/2015 17:41
File name:
Open
Cancel

# Developer Guide

This guide will go through all the classes used in this applications and will also explain the underlying functions for each class.

### cw2
This is the main entry point of the application it also contains the logic for parsing the command line arguments.

## Reader Class

This class deals with the reading from a file.

#### __init__
This initialises the reader with the file that is needs to be read which is passed in to the function as argument.

#### read_line
The calls the readline function which is applied to the file.

## JsonParser Class

This class deals with parsing the JSON file into an array

#### __init__
This initialises the array in which the data will be stored

#### add
This is used to add data to the array, it is supplied by an object which should be a dictionary

#### get_all
This is used to retrieve the list which contains all the stored data.

## Histograms Class

This class deals with creating histograms be it embed into the GUI or independent of the GUI.

### __init__

This initialises a histogram, depending if independent is true or false this would return independent chart of a chart that can be embedded to the GUI. This uses the title to set the title of the chart and the data to populate the chart. The data is a dictionary with the bins as keys and the frequency as values.

## GUI class

This class deals with the main logic related to the GUI.

### __init__

This initialises the GUI with the core elements, takes in data and stores it so that it can be used with all the queries

### find_users_by_task

This is used to find all users and sets the options of the option box with those values based on if the task requires a selected user id.

### find_users_by_doc

This is used to find the users who read a certain document and sets the options of the option box with those values when a document ID has been selected.

### btn_click_file

This is used to handle the open file button click by showing a dialog box which is used to browse for a file.

### btn_click_search

This is used to handle the search button click by calling the TaskManager.handletask method giving it the values supplied by the option boxes in the GUI

## TaskManager Class

This class represents the core functionality of the application and handles all the logic that deals with the data. This is a class with static functions that can be called using the classname.functionname style.

### get_all_users

This is used to find all the users of a supplied list of dictionary objects.

### get_all_documents

This is used to find all the documents of a supplied list of dictionary objects.

### get_all_documents_by_user

This is used to find all the documents for a certain user, which is supplied for a supplied list of dictionary objects.

### get_all_users_by_doc

This is used to find all the users for a certain document, which is supplied for a supplied list of dictionary objects.

### load_file

This is used to load file given a filename.

### task_handler

This is used to handle all task queried by the user and calls the underlying functions, it is supplied with the document id, user id, task id, data, GUI object, and if it is a command line interface of not.

### filter_data

This is used to filter a suppled list of dictionaries by the supplied key and key value. Returns a list of dictionary objects where the key value matched.

### inverse_filter_data

This is used to filter a suppled list of dictionaries by the supplied key and key value. Returns a list of dictionary objects where the key value did not match.

### get_top_10

This is used to find the top 10 readers based on their read time, it is supplied with a list of dictionary objects and returns a list of the top 10 user IDs corresponding to those users, which is ordered in a descending order.

### simple_get_all_browser

This is used to find how frequently each browser has been used by extracting the browser from the user agent string. It is supplied with a list of dictionary object and returns a dictionary with the browsers as keys and their frequency as values.

### get_all_browser

This builds on the simple version by extracting the browser name from the and ignoring the version number as it was verbose. . It is supplied with a list of dictionary object and returns a dictionary with the browsers as keys and their frequency as values.

### get_countries

This is used to find out how frequently a certain document was visited by users from a certain country. It is supplied with a list of dictionary objects and a document ID and returns a dictionary with the countries as keys and their frequency as values.

### get_continents

This builds on the get_countries by finding the continents of the users who visited a certain document. It is supplied with a list of dictionary objects and a document ID and returns a dictionary with the continents as keys and their frequency as values.

### task5

This is used to find the document that other users have read based on them reading a document a user read, this is used for suggestions. It is supplied with a list of dictionary objects, the document ID that the users has read, the user ID of the user and a high order function which is used for sorting. It returns a list of documents that other users who read this document has read.

### sort_by_readership

This is used to sort the also liked documents by the time spent reading each document. It is supplied with a dictionary and returns a sorted list in descending order of the document ID based on the time spent reading.

### sort_by_number

This is used to sort the also liked documents by number of users who read the same document. It is supplied with a dictionary and returns a sorted list in descending order of the document ID based on the number of users.

### plot_figure_gui

This is used to embed a Chart into a GUI. It is supplied with the GUI and the Chart that requires embedding.

### load_list

This is used to embed a listbox into the GUI and set the items to the list of data supplied.

## Testing

| Test Case | Expected Result | Observed Result |
|---|---|---|
| Entering invalid data in the GUI | Error message pops up | Error message pops up |
| Entering invalid task in the command line interface | Print Invalid task in  command line | Print Invalid task in  command line |
| Entering invalid document id in command line | Returns empty list and asks for valid document id or just asks for a valid document id based on task | Returns empty list and asks for valid document id or just asks for a valid document id based on task |
| Entering invalid user id in command line | Returns empty list and asks for valid user id or just asks for a valid user id based on task | Returns empty list and asks for valid user id or just asks for a valid user id based on task |

# Conclusion

I am proud of the user interface design, the functionalities that my application supports. Using high order function assisted in simplifying sorting the also likes documents, which helped  in reducing repeated code and helped in achieving more efficient code. I would have wanted to add a scroll bar to the list box in the user interface. I would have liked to add a flask version of the application and build a better interface with HTML and CSS and a Javascript library for plotting the charts but due to time constraints that was not possible.