



# F21FC Industrial programming

COURSEWORK 2 REPORT

CONTRIBUTION: H00128802(50%), H00341619(50%)

# 1. Introduction

This report details the design, implementation, testing and discussion of an application that analyses and displays document tracking data. The program aimed at completing tasks on huge datasets, including statistics graph of user information of country and browser, and liked function. For 3 million lines size datasets, the program should finish each task within 1 minute. The data is in JavaScript Object Notation (JSON). The application is developed on Python 3.8 on PyCharm 2021.2.1 (PY-212.5080.64) on Window10 (21H1 19043.1348) and designed to run on Linux platform (Ubuntu 20.04).

The program is designed to be used in Command Line Interface (CLI) ran from a Python script or a standalone executable. A Graphical User Interface (GUI) of the application is available to be launched from CLI.

We assume that there is no incorrect data in the given datasets.

## 2. Requirements' checklist

| Requirements               | Status    |
|----------------------------|-----------|
| Python 3 & Ubuntu 20.04    | Completed |
| Views by country/continent | Completed |
| Views by browser           | Completed |
| Reader profiles            | Completed |
| "Also likes" functionality | Completed |
| "Also likes" graph         | Completed |
| GUI usage                  | Completed |
| Command-line usage         | Completed |
| Standalone executable      | Completed |

## 3. Design Considerations

### Code Structure

Each task has its own class which makes the program flexible and modular. Future tasks implementations should be straightforward with this setup.

There are some shared packages, which significantly increase the reusability of the program:

-FileIO package is the low-level reading file method, chunks(), it read the json file line by line, but return chunk by chunk. Every task will need this method to read files.

-Feed\_ThreadPool package contains a method, feed\_json\_into\_ThreadPool(), that get the return dataset chunk from FileIO package and pass them into the ThreadPoolExecutor. Due

to the method also accepts a specific function that shows how to process data; hence, each task can pass their own function into the same method. The return value is a dictionary that contains desired query results.

-ThreadPool package is where actually process the given chunks. It will start the number of threads equals to the CPU on the computer, but this thread pool is not like Java or other language, since there is a Global Interpreter Lock between the interpreter and program thread, this thread pool cannot increase the processing speed.

-Plot\_Barchart package is designed for Task 2 and 3 since both have to use Matplotlib to plot graph. And the method plot\_histogram() is a decorator that aimed at add extra function to the original task. The spinner() function is to use another thread to show a spinner while analysing the data. The wrap\_function() also record the duration for analysing and plotting graph.

-Nothing\_exception package has a class Salty\_Exception which inherit from Exception class, this custom exception will catch the result contains nothing. And this exception will be used at task 2 and 3, since if there is nothing in the result dictionary, we do not have to plot a graph. We cannot apply this exception to task 4 and 5, due to that we still have to return string information to GUI part.

-TopN package has a top\_N() method that return top n items in the given dictionary, due to two tasks require return top 10 items, and maybe more tasks will require other number of top items, so, it is a good idea to make this as separated method.

We almost extract every common part from each task; therefore, it is easy to maintain and expand the functionalities. Each task is independent, and they will handle neither file nor thread pool, the task can only focus on what they should do to the data, which meets the requirement "Low coupling, high cohesion".

## Data structure

Due to the output is in two-dimension figure, the dictionary is the best choice for task 2, 3, and 4, but maybe not the best choice for task 5 and 6, which depends on the situation that average of how many kinds of document per reader. The reason is that for task 2, 3, and 4, one key has only one value, in this situation, the time complexity of create and read is  $O(1)$ , after calculation hash key, we can directly find the only value by matching the hash key. However, in task 5 and 6, one user will relate to one or more document. If there are too many value under a same key, the data structure will turn from a linked list into a red-black tree, and the complexity of create and read will become  $O(\log N)$  rather than  $O(1)$ . If the average number of kinds of document read by a reader is small, then dictionary is a good choice. Moreover, in the task, we do not want to count a same reader read a same document many times, so, the Set is very suitable, it can prevent duplicate value and we can combine Sets and a dictionary together, the value of each key in the dictionary is a set.

## Multi-Threading

We've chosen to implement the program using multi-threading to reduce "freezing" of the GUI while a task is being performed. However, due to the Matplotlib does not support multithreading, we can only apply multithreading to task 4, 5, and 6 when using the GUI, which is the user can start task 4, 5, and 6 at the same time, without waiting for a task finish and start another task.

## Data Input Batch

Due to the potential large size of the input JSON datasets (100k – 5M lines). Input is read and processed in batch (10k lines) at a time. This reduces the size of memory needed to hold the dataset.

## Error and Exception Handling

There are various error and exception handling implemented in the program, especially for user input. Due to the complex inputs, there are more error and exception handling compared to CW1. For task 2 and 3, plotting a graph is time-consuming, so, if there is nothing in the result dictionary, we want to tell the user and skip plotting the chart. For task 5, there are two main situations, one is the given user id is correct, another is no given user ID, or the given user id is incorrect. For the former one, if the given user is the only user that read the given document, then task 5d sorting will not work, because the sorting is based on also liked user. For the latter situation, if there is only one reader for the given document, we can still perform sorting tasks, because the only reader can be an also liked reader. All in all, for any tasks, if the return dictionary is empty, this will stop the task.

## Help function

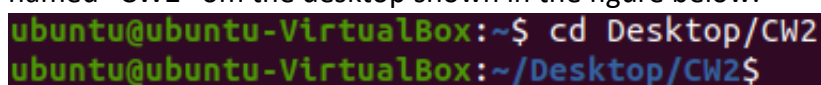
The program offers a help option which for both CLI and GUI usage. For command line, typing "-h" or "--help" will display input options in the terminal. The GUI has a dedicated button which shows the search instructions on the display window when pressed.

# 4. User Guide

## Command Line usage

The program is designed to run in Ubuntu 20.04. Before running the application, the Ubuntu system will need Python 3 and all the libraries used installed. Below are the steps to run the program in CLI or terminal as is called in Ubuntu.

1. Navigate to the project folder containing the python file. In this case it's a folder named "CW2" on the desktop shown in the figure below.



```
ubuntu@ubuntu-VirtualBox:~$ cd Desktop/CW2
ubuntu@ubuntu-VirtualBox:~/Desktop/CW2$
```

Figure 1 - Screenshot of directory navigation.

2. Once in the folder directory, run the python script with input options as in the format shown in the figure below:

```
:~/Desktop/CW2$ python3 cw2.py -f file_name -d doc_uuid -u user_uuid -t task_id
```

Figure 2 - Screenshot of running the program with input options.

In our case, the main python script file is called “cw2.py”. The input options are:

- **-f file\_name:** is the name of the .json file that contains the dataset. Replace “file\_name” with the name of the file (including .json file extension).
  - **-d doc\_uuid:** is the document ID number that will be searched. Replace “doc\_uuid” with the ID number of the document desired.
  - **-u user\_uuid:** is the user ID number that will be searched. Replace “user\_uuid” with the ID number of the user desired.
  - **-t task\_id:** is the task ID number that will be performed. Each task required by the coursework specifications are assigned its own number. Replace “task\_id” with the ID number of the task desired. Task IDs and their functions are listed in *table 1*.
3. Every task requires a JSON filename. Task 2a, 2b, 5d and 6 requires a document ID. Task 5d and 6 can require a user ID. Tasks that generate graphs will open a new window displaying the requested graph. Tasks that output text will be shown in the terminal.
  4. A help function is available by writing “--help” after the python script name as shown in figure 3.

```
ubuntu@ubuntu-VirtualBox:~/Desktop/CW2$ python3 cw2.py --help
#####
options:
  -u --userID: input the userID that need to query (Optional)
  -d --docID: input the documentID that need to query
  -t --taskID: input one of the task that want to run, [2a, 2b, 3a, 3b, 4, 5d, 6, 7]
  -f --filename: input the JSON file that contains the data
  -h --help: ask for help
  -q --quit: exit immediately
#####
```

Figure 3 - Screenshot of the help function.

5. A standalone executable is also available to run in CLI (*figure 4*). The process is mostly the same as in step 2 except python interpreter is not need for the executable. If in the file directory of the executable, simply type: “./filename” (replace “filename” with the name of the executable) followed by the same input options as before.

```
ubuntu@ubuntu-VirtualBox:~/Desktop/cw2_standalone$ ./cw2 -f 100k.json -d 100806162735-00000000115598650cb8b514246272b5 -u 00000000deadbeef -t 4
```

Figure 4 - Screenshot of the executing a standalone executable with real input options.

| Task ID   | Task Performed   |
|-----------|--|
| <b>2a</b> | Display histogram of country/countries and the number of occurrences for a given document.   |
| <b>2b</b> | Display histogram of continent/continents and the number of occurrences for a given document.  |
| <b>3a</b> | Display a histogram of all browser identifiers of the viewers.   |
| <b>3b</b> | Display a histogram of main browsers name of the viewers.  |
| <b>4</b>  | Outputs the top 10 readers determined by the total user reading time.  |
| <b>5d</b> | Outputs the “also likes” of a given document. The “also likes” functionality identifies which other documents are also read by the reader of the given document. |
| <b>6</b>  | Display a histogram of the “also likes” function.  |
| <b>7</b>  | Opens A GUI window which can be used to complete all the above tasks. Task 6 is also performed when opening the GUI as required by coursework specification.     |

Table 1 - Table of Task ID and their functions.

## GUI Usage

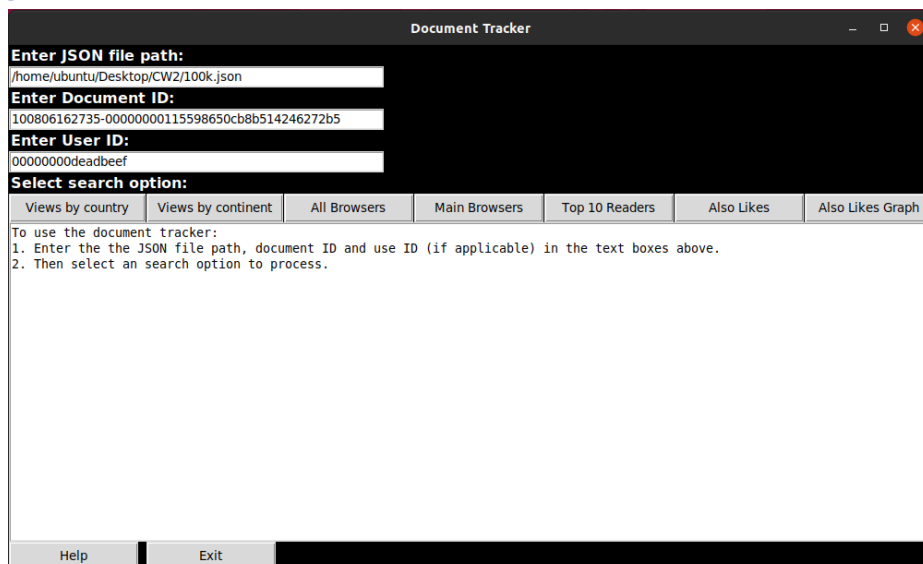


Figure 5 - Screenshot of GUI running.

To launch the GUI, type “-t 7” in the command line interface as described in CLI usage.

The GUI is made up of three input fields each for the file path, document ID and the user ID as shown in *figure 5*. Below the input fields are the buttons which performs tasks from 2a to 6. The output textbox takes up most of the centre of the GUI. A help and exit button exist at the bottom of the window. Pressing any search buttons will perform that task with the file path and IDs entered.

Tasks that plot graphs will open an external window for the graphs. Tasks that return text will be displayed in the output textbox.

When launched, it will take inputs from the CLI and copy it into the input field of the GUI automatically and perform task 6 as required.

## 5. Developer Guide

### Libraries

Libraries used in this program:

- json library is used for processing the JSON datasets
- pandas library is used to read in the country/continent csv file for the dictionary.
- Matplotlib library is used to plot charts for output.
- Tkinter library is used to build the GUI for the program.
- Graphviz package is used to display .dot format graphs and conversion to .pdf format.

### Package - cw2

This file mainly provide function to achieve CLI interactions. The reason using `sys.argv[1]` is that the zero position is the name of the file.

```
opts, args = getopt.getopt(sys.argv[1:], 'u:d:t:f:hq', ['userID=', 'docID=', 'taskID=', 'filename=', 'help', 'quit'])
```

in the for loop, there are some file name and type check, as well as task id check. If there are more tasks in the future, they can be easily added in the if-elif chunk at the bottom.

### Package - FileIO

fileIO file contains only one methond chunks(), the size can be changed according the size of memory. This method is only used in feed\_json\_into\_Threadpool() method in Feed\_ThreadPool package. Yield keyword helps us to have a save point, we do not need to maintain an extra variable that record the number of lines. The reason that doesn't have to use multithreading here is it is not a compute intensive task, we only need to read the data from the storage device and load into memory, so, the bottleneck of speed is how fast the storage can read.

## Package - Feed\_TreadPool

feed\_json\_threadpool file contains only one method feed\_json\_into\_Threadpool(), each task will call this method directly. process\_method parameter is the way how to process data, this function return nothing rather sharing a same result dictionary, since we only pass reference into this method rather than the value. In this method, we should also check the file name and type again, because when using the GUI, there is no file exception checking.

## Package - ThreadPool

Threadpool file contains one method threadpool\_execute(), in this method, we will create many subtasks, which the only difference is each thread will process different part of the dataset. So, in args, we must use list comprehension to create a long tuple. In the map higher-order function, we must unzip those tuples first, so that the process method can recognise them. Here, it's not necessary to divide by 8, it depends on the number of cores in the CPU.

```
args = ((lock, userID, docID, dataset[i:i + len(dataset) // 8], result)
        for i in range(0, len(dataset), len(dataset) // 8))
executor.map(lambda f: func(*f), args)
```

if the size of dataset cannot divide by 8 exactly, then we can believe the size must less than 10k, and it is small enough to be a single task for thread pool.

```
executor.submit(lambda f: func(*f), [lock, userID, docID, dataset, result])
```

But why we still need a lock here? Although there is GIL to make sure only one thread will work at a same time, this can only guarantee that only one line bytecode will be executed, not one line in the program. Since one line code will be interpret into one or more lines bytecode, so, we still need lock to explicit make sure there is no competition or dirty data.

```
lock = threading.RLock()
```

## Package - TopN

topn file contains only one method top\_N(), using a priority queue to get top n item is a very common method in competitive programming. If the number of top item is much more less than the size of data, we can believe it only has a linear time complexity  $O(N\log M)$ , where N is the size of data, M is the number of top items. In our case, N is around million, but M is only 10, so we can retrieve the top 10 items in a short time.

## Views by Country/Continent

Since only country code are represented in the JSON dataset. We used a dictionary which holds all countries and their information relative to the country code. When searched for a specific document ID, this dictionary is used to store the count of each country code after matching the document ID and event type. In task 2a, we can simply represent the result dictionary, but in task 2b, we need a further process to replace the country code into continent. Here, we used an external source, all.csv, which contains each country code and its continent. So, we extract that information into a new dictionary, the key is the country code, the value is the continent. Finally, we can replace the country easily.



```

table = pd.read_csv(os.path.abspath(r"Task2/all.csv"))
country_to_continent = table.set_index('alpha-2')['region'].to_dict()
.....
# replace the country to continent
for x in results:
    new_results[country_to_continent.get(x)] = results.get(x)

```

## Views by Browser

Task 3 is very similar to task 2, the only difference is how to handle the user-agent. Due to historic reason, many browsers will disguise itself as other browser to get a higher quality code from servers. That is the reason why there will be many browser names in one user-agent string. However, we can still extract the real browser name using different priority filter. For example, Chrome tends to pretend as Safari, but the Safari never pretend as any other browser. In this case, we can check whether there is a “Chrome” substring in the user-agent first, and then, check whether there is a “Safari” substring.

```

elif Chrome.search(string):
    answer = "Chrome"
elif Safari.search(string):
    answer = "Safari"

```

## Top 10 Reader Profiles

This function returns the top ten readers by total reading time of each user. Top readers are saved and are displayed with their total reading time. In this task, we can group "pagereadtime" event type by user ID, and using TopN package to get top 10 reader.

## “Also likes” functionality

There are three steps to achieve this function. First is to build the user and document relationship, one user to one or more documents, and stored in a dictionary. Second is to record which user read the given document ID in a set while we are processing the data. Finally, simply filter the result dictionary using the set. The final result is only those reader who have read the given document will left, no matter the user ID is given or not.

```

def also_likes(userID, docID, filename):
    global user_set
    global temp_display
    results = feed_json_threadpool.feed_json_into_Threadpool(userID, docID,
filename, process_method)
    if not (userID in user_set):
        temp_display = "Given user ID is not in the also likes user list,
and will be ignored\n"
        # get the users that read docID
        for u in user_set:
            new_results[u] = results.get(u)

```

Due to task 5d require a default sorting algorithm, which is top 10 also liked document, again, we just apply topN function here. However, we need to get the count for each document first, there are two possible situation which we have discussed in design consideration.

```

if not (userID in result):
    for x in result:
        for d in result.get(x):

```

```

        count[d] = count.get(d, 0) + 1
else:
    del result[userID]
    for x in result:
        for d in result.get(x):
            if d != docID:
                count[d] = count.get(d, 0) + 1

```

## “Also likes” graph

When building the graph, just like inorder traversal of a n-ary tree, the father node first be printed, then each of its child nodes be printed. Pseudocode provided.

```

for each father node:
    print father node
    for each its child node:
        print child node

```

## GUI

The GUI is a simple window containing rows and columns. Input text fields, labels, buttons and a output textbox makes up of all the components in the GUI. Each GUI button has a command which is linked to a specific function. For example:

```
Button(window, text="Views by country", width=15, command=click_country)
```

This creates a button which is for views by country and has the click command which runs the function for processing task 2a.

Each task processing function will read in the inputs from the input text fields which contains the file path, document ID and user ID and passes to the relevant task classes for processing. Once request is processed, it will output the results in the output text box.

## 6. Testing

### Errors and Exceptions Testing Table

| Tests                                      | Outcome   |
|--|---|
| General                                    |   |
| Wrong file name                            | Output "Cannot find file", filename, "because it does not exist." |
| Wrong file format                          | Output "Wrong file type:", filename, "not a json file."           |
| File access denied                         | Output "Cannot open the file ", filename                          |
| No input file name                         | Output "No file input!"   |
| No input task id                           | Output "No given task ID!"  |
| Input wrong task id                        | Output "Unknown task ID", opt_value                               |
| Task 2                                     |   |
| Input wrong docID or not found given docID | Output "Nothing found in this file!", filename                    |

|   |  |
|---|--|
| Task 3  |  |
| Not found any reader  | Output "Nothing found in this file!", filename                                     |
| Task 4  |  |
| Not found any reader  | Output "Not found any readers in the file" + filename                              |
| Task 5  |  |
| No input user ID or user ID not correspond to the given document ID | Output "Given user ID is not in the also likes user list, and will be ignored"     |
| Not found given document ID   | Output "There is no user read this document " + docID + " in the file " + filename |
| Only one reader for given document ID                               | Output "There is only one user read this document!"                                |
| Task 6  |  |
| No input user ID or user ID not correspond to the given document ID | Output "Given user ID is not in the also likes user list, and will be ignored"     |
| Not found given document ID   | Output "There is no user read this document " + docID + " in the file " + filename |
| Only one reader for given document ID                               | Output "There is only one user read this document!"                                |
| Task 7  |  |
| No input file name  | Output "No filename entered"   |
| No input document ID  | Output "No doc ID entered"   |
| Wrong file name   | Output "Cannot find file", filename, "because it does not exist."                  |
| Wrong file format   | Output "Wrong file type:", filename, "not a json file."                            |

## Performance Testing Table

| File Size       | Average Execution Run-time (s) |
|-----------------|--------------------------------|
| 100k lines      | 0.5                            |
| 400klines       | 4.3                            |
| 600k lines      | 6.4                            |
| 3 million lines | 29.2                           |

Performance tests was ran in a virtual machine (Ubuntu 20.40) on a laptop with an i7-9750H CPU. For each file size, the runtime of each task was recorded, and the average is shown in the table.

## 7. Programming Language and Implementation

### Python Features and Technologies

Python has high readability and simple syntax. It is designed to be easy to learn. It does away with curly braces and end statements, instead it uses indentation. Variables doesn't have to be explicitly defined by the user and can be left to Python to decide. Another feature of Python is that it 'interpreted' instead of 'compiled'. When comparing the two, Beri describes that "Interpreted languages have a major advantages such as they are portable, which means they can run on different operating systems and platforms." (Beri, 2019) . Python also has a large collection of libraries, which makes it very suitable for machine learning.

For advanced language features, we found the decorator is very useful, one function can have one or more decorators, and they will be executed from bottom to top. We can use "@" this syntactic sugar to call a decorator, and the decorator won't affect the original function. The higher-order functions are very pythonic, one line can solve most problem, if not, then two lines. Generator comprehensions is another way to increase pythonic, we do not need an extra list to store a temporary value generated by for loops rather we can pass the loops into where we need it. When read from json files, we use yield which will return a generator, for each query, the program will start from yield key word again until read over. So, we can read chunk by chunk and will not forget where we stopped last time, this is kind of lazy data structure. ThreadPoolExecutor should be a very helpful library if there no GIL, we can use map() or submit() to give many tasks or one task to the pool. Technically, a thread pool can speed up many times than single thread according to the experience in Java.

Some limitations to Python are its performance compared to 'compiled' languages. When compared to C# from coursework 1, C# is very similar to other popular languages such as C, C++, and Java it is more familiar for programmers coming from those languages. Python is more unique and takes a little longer to get used to when starting.

Another downside is that python does not support "true" multi-threading concurrency due to the Python Global Interpreter Lock (GIL). This lock only allows one thread to process the Python interpreter at a time which bottlenecks systems with multiple CPU threads (Lutz, 2010, p. 211). A solution to this is to use the multiprocessing package which offers concurrency bypassing the GIL by using multiple processes rather than threads (Badhwar, 2018, p. 90). But currently this is only available on Unix and Windows (Foundation, 2021).

Moreover, when programming python, we always must keep the data type of each variable in mind. And having to check whether the data type is match from time to time. Therefore, a good habit that give an easy to know name of variables is very important. In system language, the compiler will do this for programmer.

## Document Tracker

For an application such as the document track created in this part of the coursework, various libraries were used. This makes Python suitable thanks to its library availability.

Our program only processes 3 million lines of JSON dataset. In a real-world scenario, the datasets will be much larger therefore will take a substantial amount of time to process using our application. And of course, tech companies won't use this way to analyse data rather using database. Assuming we still want to break the linear time base line, one possible solution is to build index, since user id and document id is unique. We can use B tree or B+ tree to record the ID value and other information, although we must have to use extra to store those index files. Depends on how to build index, the time complexity can become  $O(\log N)$  or even  $O(1)$ .

Due to the GIL, implementation in Python will be slower than that of other languages that can take advantages of multi-threading to improve performance. This performance difference will be especially clear on larger JSON input files as processing time increases.

## 8. Discussion from CW1

Most of the negative feedback from CW1 stems from code quality and report writing.

### Coding

The aim of this program was to have nicely structured code and readable. Each task is split into its own class to allow better structure and navigation. Suitable data structures were chosen for the tasks. Repeatable codes were reduced. Performance of the application is also considered and recorded. We have to know the time complexity of CRUD of each data structure, since these are very popular question in coding interview. We should also improve algorithm knowledge, at least those popular algorithms. Next is to learn how to design a better structure of a large project.

### Report

The report is kept concise and understandable. Each section is labelled correctly with figures and tables to aid understanding. Testing is now more thorough and covers more errors and exceptions. Developer guide is more detailed this time. More insight has been added in the discussion for language reflection and implementation.

## 9. Conclusion

In conclusion, we are satisfied with the overall implementation of the program. All specifications were met, and the application was bug-free from our testing.

Improvements can be made to improve the performance of the program by implementing "true" multithreading. This can be achieved on a different language or on a different system that supports multiprocessing.

## References

Badhwar, S., 2018. *Hands-On Enterprise Application Development with Python: Design Data-intensive Application with Python 3*. s.l.:Packt Publishing.

Beri, R., 2019. *Python Made Simple: Learn Python Programming in Easy Steps with Examples*. s.l.:BPB Publications.

Foundation, T. P. S., 2021. *multiprocessing — Process-based parallelism*. [Online] Available at: <https://docs.python.org/3/library/multiprocessing.html>

Lutz, M., 2010. *Programming Python: Powerful Object-Oriented Programming*. s.l.:O'Reilly Media.