Assessed Coursework 2 — Data Analysis of a Document Tracker

1 Overview

The aim of this coursework is to develop a simple, data-intensive application in Python 3.

This is an **pair project**, and you will have to identify the contributions of each group member in the report. Each group member needs to contribute a substantial implementation task to the project.

The **learning objective** of this coursework is for students to develop proficiency in advanced programming concepts, stemming from both object-oriented and functional programming paradigms, and to apply these programming skills to a concrete application of moderate size. Design choices regarding languages, tools, and libraries chosen for the implementation need to be justified in the accompanying report.

This coursework will develop personal abilities in using modern scripting languages as a "glueware" to build, configure and maintain a moderately complex application and deepen the understanding of integrating components on a Linux system.

The report needs to critically reflect on the software used for implementing this application, and discuss advantages and disadvantages of this choice. The report should also contain a discussion, contrasting software development on Windows and Linux systems and comparing software development in scripting vs. systems languages (based on the experience from the two pieces of coursework).

2 Lab Environment

Software environment: You should use Python 3 as installed on the Linux lab machines (EM 2.50) for the implementation. This installation also provides the `pandas`, `tkinter`, and `matplot` libraries.

If you want to develop the software on your own laptop you need to install the above software. Both Python and the libraries are available for download at: https://www.python.org/download.

For each of the chosen technologies, the report should discuss why it is the most appropriate choice for this application, and possible alternatives should be mentioned.

3 Data Analysis of a Document Tracker

In this assignment, you are required to develop a simple Python-based application, that analyses and displays document tracking data from a major web site.

The issuu.com platform is a web site for publishing documents. It is widely used by many on-line publishers and currently hosts about 15 million documents. The web site tracks usage of the site and makes the resulting, anonymised data available to a wider audience. For example, it records who views a certain document, the browser used for viewing it, the way how the user arrived at this page etc. In this exercise, we use one of these data sets to perform data processing and analysis in Python.

The data format is described on this web site http://labs.issuu.com/dataset_spec.html (see also this local page for the data spec). Familiarise yourself with the details of the data representation before you start implementation. The initial (small) data set used for this exercise is available online here: http://www.macs.hw.ac.uk/~hwloidl/Courses/F21SC/issuu_cw2.json. For testing use this tiny sample data file at http://www.macs.hw.ac.uk/~hwloidl/Courses/F21SC/issuu_sample.json.

The application needs to run on an up-to-date Linux platform (Ubuntu 18.04). The application should be developed in Python 3, using appropriate libraries for input, data processing and visualisation. Possible

choices are the `json` library for parsing, the `pandas` library for processing the input data (optional), the `tkinter` library for GUI functionality and the `matplot` library for visualising the results. You need to identify the advantages of your choice of libraries.

The application must provide the following functionality:

1. **Python:** The core logic of the application should be implemented in Python 3.

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 `matplotlib`.

   (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 `matplotlib`.

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.

   (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.

4. **"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.

   To achieve this task you will need to do the following:

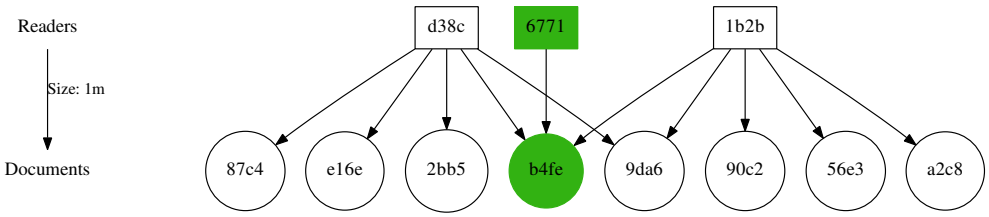
   (a) Implement a function that takes a document UUID and returns all visitor UUIDs of readers of that document.

   (b) Implement a function that takes a visitor UUID and returns all document UUIDs that have been read by this visitor.

   (c) Using the two functions above, implement a function to implement the "also like" functionality, which takes as parameters the above document UUID and (optionally) 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, and use the sorting function parameter instead.

(d) 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*. Provide a document UUID and visitor UUID as input and produce a list of top 10 document UUIDs as a result.

5. **"Also likes" graph:** For the above "also like" functionality, generate a graph that displays the relationship between the input document and all documents that have been found as "also like" documents (and only these documents). Highlight the input document and user by shading in that graph, and use arrows to capture the "has-read" relationship (i.e. arrow from reader to document). In the graph shorten all visitor UUIDs and document UUIDs to the last 4 hex-digits. As an example, the graph below uses document `b4fe` and reader `6771` as input (shaded green) and displays 7 "also like" documents, together with the readers that relate these documents with the input document (download `.dot` source file and `.ps` output file):



Hint: Use the `.dot` format as graph representation. Use the `graphviz` package with the `dot` tool to translate the `.dot` into a `.ps` format (and then optionally in `.pdf` format). For a detailed description see this `dot` User Manual. You can install the `graphviz` package on an Ubuntu machine by typing (in a terminal window): `sudo apt-get install graphviz`

As an example of `graphviz/dot` usage, you can generate the above graph from the `dot` source code and view it like this:

```
% wget http://www.macs.hw.ac.uk/~hwloidl/Courses/F21SC/1m_3.dot
% dot -Tps -o 1m_3.ps 1m_3.dot
% evince 1m_3.ps
```

6. **GUI usage:** To read the required data and to display the statistical data, develop a simple GUI based on `tkinter` that reads the user inputs described above, and with buttons to process the data as required per task.

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 -f file_name
```

to check the results of implementing task `task_id` using inputs `user_uuid` for the user UUID and `doc_uuid` for the document UUID; `file_name` is the name of the JSON file with the input data. The `task ids` should be: `2a, 2b, 3a, 3b, 4d, 5, 6`, matching the tasks above (task id `6` should run Task 5 and then automatically launch a GUI showing the resulting also-likes graph).

4 Submission

You must submit the **complete project files, containing the source code, a stand-alone executable, and the report** (in `.pdf` format) as one `.zip` file no later than **3:30 PM on 5th of December 2019**. Addition-

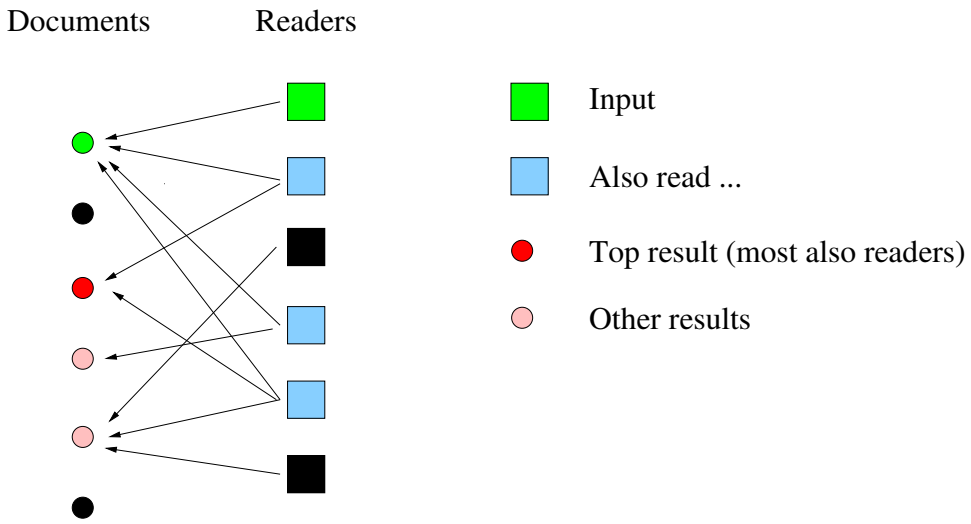


Figure 1: Example of identifying also-likes documents. Starting from the current reader and document (green), all readers are identified, who have also read the input document (blue). From the other documents, read by these readers, the top 10 documents, counted by number of readers are identified and displayed. In this example the red document is top of this list, and the two pink documents are also on the result list. The automatically generated graph should display all three result documents, but doesn't have to distinguish between "best" and "others" by shading. The unused, black users and documents shouldn't be shown in that graph.

ally, a **screencast or video** of running the application, with an explanation as voice-over audio needs to be submitted to Vision. This is mandatory, and without the screencast or video the submission is incomplete and may be marked as 0 points. The standard penalty of -30% of the maximum available mark applies to late submissions. No submissions will be accepted after 5 working days beyond the submission deadline. The main function driving the application should be called `cw2`, as discussed in "Command-line Usage" above. Submission must be through *Vision,* submitting all of the above files in one `.zip` file. Additionally, an electronic submission of report, sources and application by email is recommended. **This coursework is worth 35% of the module's mark.**

You are marked for your application, the structure, code and comments used, your testing, your report and the screencast/video of demonstrating the running of your application. The marking scheme for this project is attached.

5 Report Format

The report should have between 10–20 pages and use the following format (if you need space for additional screenshots, put them into an appendix, not counting against the page limit, but don't rely on the screenshots in your discussion):

1. **Introduction:** State the purpose of the report, your remit and any assumptions you have made during the development process.

2. **Requirements' checklist:** Here you should clearly show which requirements you have delivered and which you haven't.

3. **Design Considerations:** Here you should clearly state what you have done to your application to make it more usable and accessible.
4. **User Guide:** Use screen shots of the running application along with text descriptions to help you describe how to operate the application.
5. **Developer Guide:** Describe your application design and main areas of code in order to help another developer understand your work and how they might develop it. You may find it useful to supplement the text with code fragments.
6. **Testing:** Show the results for testing all cases and prove that the outputs are what are expected. If certain conditions cause erroneous results or the application to crash then report these honestly.
7. **Reflections on programming language and implementation:** Based on your experience in implementing this application, reflect which language features and technologies have been most helpful, identify limitations of your application and suggest ways how to overcome this limitations. Also reflect on the usability of the (kind of) language (either system or scripting language) for this application domain, and on its wider applicability.
8. **What did I learn from CW1?** A short discussion on lessons learnt from the feedback given on CW1 and a discussion how you integrated this feedback into CW2. Cover both coding and report writing, possibly more (project management, preparing for interview style questions etc).
9. **Conclusions:** Reflect on what you are most proud of in the application and what you'd have liked to have done differently.
10. A final section should contain the main references used in this report and in the implementation. *These references need to be cited in the report to indicate where the reference has been used in the report or implementation.*

Marking Scheme

Criteria	**Marks**
Meeting system requirements	45
Report Quality Content and communication (Design Considerations, Reflections, User guide & Developers guide, Testing)	20
The Application Code quality, secure coding, sufficient comments, sensible variable/object names, good code/class/method design, use of advanced lang. features.	15
Initiative, innovation, **professional conduct**, creativity and efforts above & beyond the call of duty	15
Screencast/Video of running the application with explanation as voice-over audio.	5
Total marks	100

Professional Conduct and Plagiarism

This is a **pair project**, and you will have to identify the contributions of each group member in the report. Each group member needs to contribute a substantial implementation task to the project. Where external resources have been use, these need to be clearly identified and referenced.

Plagiarism:
This project is assessed as an **pair project**. You will work with your partner on the implementation and should split up tasks in a reasonable way to arrive at a complete implementation of the specification. You will have to identify the contributions of each group member in the report. You can discuss general technical issues related to this work with other students, however, you must not share concrete pieces of code or text. Readings, web sources and any other material that you use from sources other than lecture material must be appropriately acknowledged and referenced. Plagiarism in any part of your report will result in referral to the disciplinary committee, which may lead to you losing all marks for this coursework and may have further implications on your degree. For details see this link

Late Submission Policy

The standard **penalty of -30%** of the maximum available mark applies to late submissions. No submissions will be accepted after 5 working days beyond the submission deadline.