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.

This is an **individual project**, and you will have to submit your own, original solution for this coursework specification, consisting of a report, the source code and an executable.

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. Familiarise yourself with the details of the data representation before you start implementation. The main data set used for this exercise is http://www.macs.hw.ac.uk/~hwloidl/Courses/F21SC/Data/issuu_cw2.json. For testing use the smaller data file at http://www.macs.hw.ac.uk/~hwloidl/Courses/F21SC/Data/issuu_sample.json.

The application should be developed in Python, 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.

Deadline: 3:30PM on Friday 2nd/Sunday 4th of December 2016; (**individual project**)

The application must provide the following functionality:

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

   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 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.

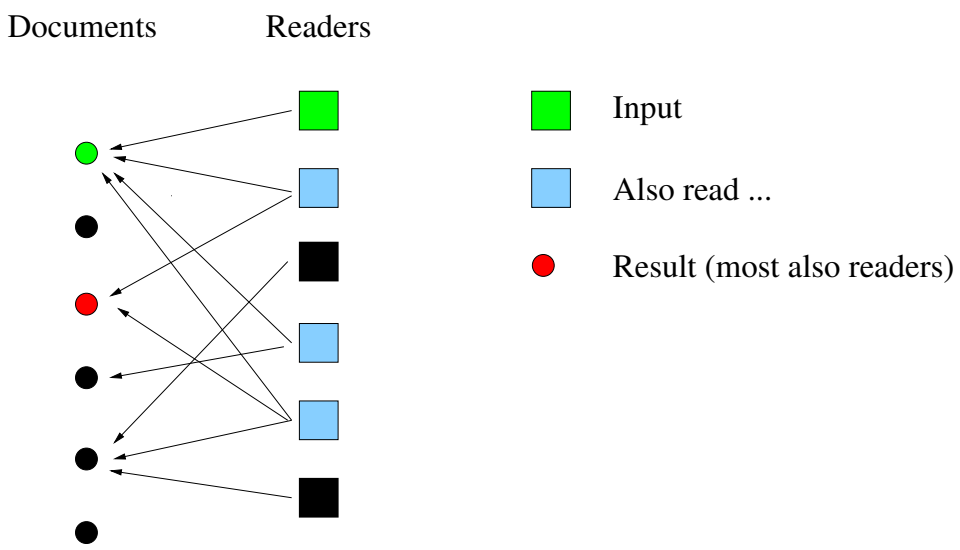

Figure 1: Example of identifying also-likes documents. Starting from the current reader and document (green), all readers are identified, who have also read the input document (blue). From the other documents, read by these readers, the top 10 documents, counted by number of readers are identified and displayed. In this example the red document is top of this list.

(d) Use this function to produce an "also like" list of documents, using the above function, based on *readership profile for sorting the documents*.

(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*.

(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.

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.

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.

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 Friday 2nd/Sunday 4th of December 2016**. For Edinburgh campus the Friday, for Dubai campus the Sunday deadline applies. 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, as well as posting a hardcopy

of the report with a cover sheet in the appropriate drop-box near the School Office 1.24. Additionally, an electronic submission of report, sources and application by email is recommended. **This coursework is worth 50% of the module's mark.**

You are marked for your application, the structure, code and comments used your testing, your report and your demonstration. The marking scheme for this project is attached. Following the submission, there will be mandatory demos, where you will have to present your implementation, explain its functionality and implementation, and you must be prepared for answering knowledge questions about the programming language and library functions.

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. **Conclusions:** Reflect on what you are most proud of in the application and what you'd have liked to have done differently. An optional final section of references is also encouraged.

Marking Scheme

Criteria	**Marks**
Meeting system requirements	50
Report Quality Content and communication (Design Considerations, Reflections, User guide & Developers guide, Testing)	15
The Application Code quality, sufficient comments, Sensible variable/object names, Good code/class/method design.	15
Initiative, innovation, creativity and efforts above & beyond the call of duty	10
Demo (presentation of implementation, knowledge questions)	10
Total marks	100