COMSC 1450 Introduction to Programming
Term Project Spring 2022
Dr. Carlos Monroy

Title: Up to you and be creative!
Due Date: See below for details

Learning Objectives: In this project, students will:

a) Implement an end-to-end programming application in Python encompassing the main topics covered throughout the semester.
b) Work in teams of two and use GitHub functionalities for version control when developing software in teams.
c) Experience the process of Conceptualization, Design, Implementation and Testing of a software application.

Deliverables:

a) A repository in GitHub named **`s22-python-proj-username`** (because you are working on teams of two, use the GitHub **`username`** of one of your team members will be designated as team manager).
b) Seven slides maximum in a pdf document, submitted via Blackboard, to be used for the final presentations.
c) Four progress reports, to ensure that the work is on track, submitted via Blackboard (both team members should submit a report with comments, be specific and include details). Progress reports dates will be indicated in Blackboard.

Due Dates:

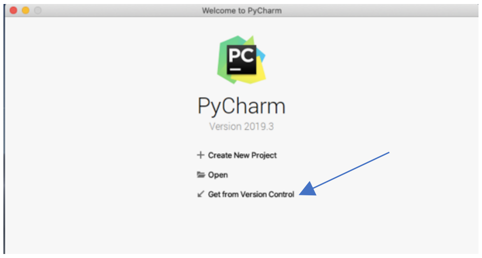
Date	Deliverable	Description
Monday April 25th	**Progress report 1**	Detailed progress report. Include screen shots on how you understand the relationships among packages, modules and variables. Each team member submit a separate report. Submitted via Blackboard in pdf format.
Friday April 29th	**Progress Report 2**	Detailed progress report. Each team member submit a separate report. Submitted via Blackboard in pdf format.
Wednesday May 4th	**Progress Report 3**	Detailed progress report. Each team member submit a separate report. Submitted via Blackboard in pdf format.
Monday May 9th	**Progress Report 4**	Detailed progress report. Each team member submit a separate report. Submitted via Blackboard in pdf format.
Tuesday May 10th 11:59PM	**Final Code**	Final commit and push submitted via GitHub, by creating a release named v.0.5.0
Wednesday May 11th	**Code walkthrough meeting**	30-minute session with each team: for walk through your code; explain your solution, and answer questions about your code. Both team members must be present. Signup link will be provided.
Thursday May 12th 10:00 AM	**PowerPoint slides**	Final slides for Thursday presentation, submitted via Blackboard
Thursday May 12th 12-2:30 PM	**Final presentation**	Each team presents their project (both team members must present).

Repo creation and cloning:

Only the team coordinator will create a repo by clicking the following link (this was already done in class), no need to redo.

https://classroom.github.com/a/foAEUha0

This will create a repo named **`s22-python-proj-username`** (where username is the team coordinator's username). Once done, both team members will clone into the local computer in PyCharm using the option Get from Version Control, in the same way we did for the other two repos (homework and examples).

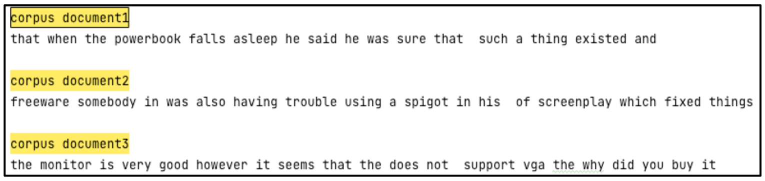


Part I. General Description

Instructions: The application is intended to gather, information and statistics about a collection of documents. The documents are text files corresponding to summaries of movies. To simplify the project the collection of documents is contained in one file.

File name	Description
fake_corpus.txt	A small file with 3 documents to illustrate a small input file, the corresponding dictionaries are located in the script: **parser/fake_builder.py**
text_corpus.txt	A small file with 3 documents to be used for developing and implementing your work, so that it is easy to verify your output.
text_corpus_large.txt	A larger file with 1,000 documents, you can use this file to test your application once it is completed.
text_corpus_larger.txt	A larger file with 2,000 documents, you can use this file to test your application once it is completed with a larger dataset.

The input text files are separated by a line with the text corpus documentX, where X is a number. Here's a screen shot of the fake_corpus.txt file, showing 3 documents.



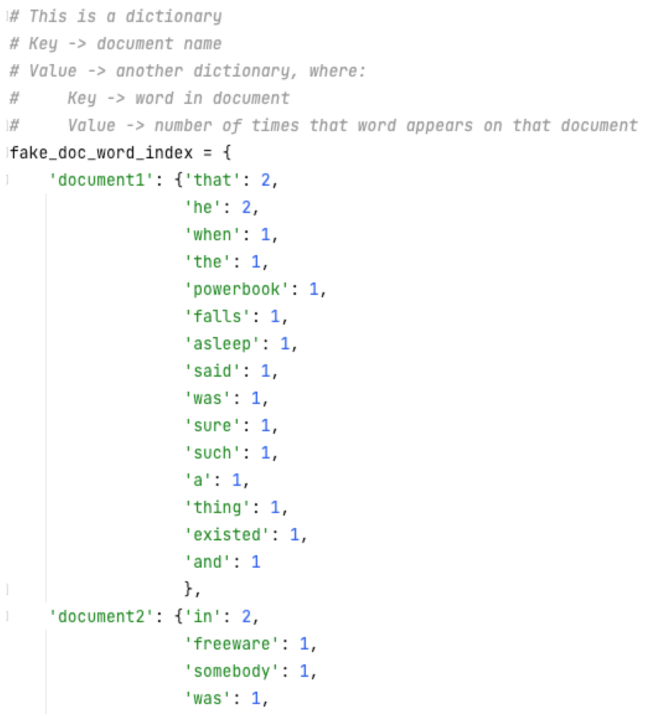
```
corpus document1
that when the powerbook falls asleep he said he was sure that  such a thing existed and

corpus document2
freeware somebody in was also having trouble using a spigot in his  of screenplay which fixed things

corpus document3
the monitor is very good however it seems that the does not  support vga the why did you buy it
```

You will write code in scripts in the package named celtic. A package in python is a bundle of scripts that provide functions and variables with some functionality, it is organized as a directory with sub-directories. The package is then broken down into modules, a module is a directory that contains one or more python scripts with functions and variables that provide some functionality. The project includes the following modules:

- parser: scripts for reading input text and break down in tokens (words), build dictionaries of words and documents from the entire corpus.
- stats: scripts for calculating statistics about each document and the entire collection. See details below.
- viz: using the turtle library to plot stats as bar graphs (this module is a bonus module).

The parser module: a parser is a collection of functions that reads text from a file or files, splits the lines into tokens (words) and builds dictionaries. Look at the script parser/fake_builder.py to see how the contents of the dictionaries should look like, given the input data in the file texts/input/fake_corpus.txt

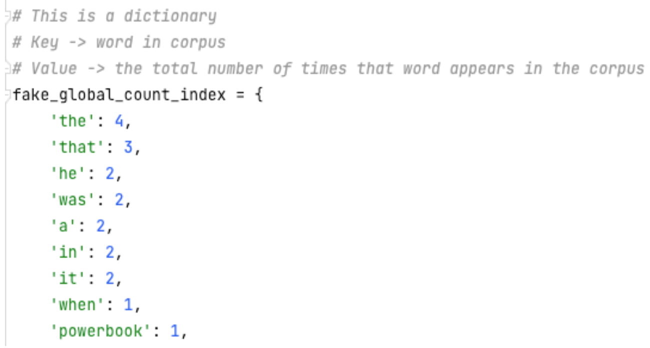
Here are a couple of examples:



```
# This is a dictionary
# Key -> document name
# Value -> another dictionary, where:
#     Key -> word in document
#     Value -> number of times that word appears on that document
fake_doc_word_index = {
    'document1': {'that': 2,
                  'he': 2,
                  'when': 1,
                  'the': 1,
                  'powerbook': 1,
                  'falls': 1,
                  'asleep': 1,
                  'said': 1,
                  'was': 1,
                  'sure': 1,
                  'such': 1,
                  'a': 1,
                  'thing': 1,
                  'existed': 1,
                  'and': 1
                  },
    'document2': {'in': 2,
                  'freeware': 1,
                  'somebody': 1,
                  'was': 1,
```

fake_doc_word_index is a dictionary, where the key is the document name and the value is a nested dictionary with a word as key, and a count as value. In this example in "document1", the word "that" appears 2 times. The word "he" appears 2 times, and the word "when" only 1 time.

In document2, the word "in" appears 2 times, "freeware" 1 time, etc.



```
# This is a dictionary
# Key -> word in corpus
# Value -> the total number of times that word appears in the corpus
fake_global_count_index = {
    'the': 4,
    'that': 3,
    'he': 2,
    'was': 2,
    'a': 2,
    'in': 2,
    'it': 2,
    'when': 1,
    'powerbook': 1,
```

fake_global_count_index is a dictionary, where the key is a word and the value is a number indicating how many times that word appears in the entire corpus. In this example the word “the” appears 4 times. The word “that” appears 3 times, and so on.

The table below provides the API (Application Program Interface), that is the functions that you have to write and implement (except the first two that have already been completed and are provided as examples). Look into the script **builder.py** to learn more about the contents of each dictionary and what is that each function should return. Unique words mean a word that appears only 1 time in a document.

Returned value is stored in variable	Functions that build dictionaries, the parameter data is a container to be use as input on each function	Description
doc_word_index	**build_doc_word_index(data)** This function is already defined and implemented, do not change it.	Builds a dictionary where the key is the name of the document, and the value is another dictionary, where the key is a word in that document and the value is a number of how many times that word appears in that document.
global_count_index	**build_global_count_index(data)** This function is already defined and implemented, do not change it.	Build dictionary where the key is a word in the entire corpus and the value is the number of times that word appears in the entire corpus.
word_count_index	**build_word_count_index(data)**	Build a dictionary where the key is the name of the document, and the value is another dictionary, where the key is a word in that document and the value is a number of how many times that word appears in that document divided by the total number of words in that document.
weighted_word_count_index	**build_weighted_word_count_index(data)**	Build a dictionary where the key is the name of the document, and the value is another dictionary, where the key is a word in that document and the value is a number of how many times that word appears in that document divided by the number of unique words in that document.
doc_inverted_index	**build_doc_inverted_index(data)**	Build a dictionary where the key is a word and the value is a list of all documents where that word appears.
doc_dictionary	**build_doc_dictionary(data)**	Build a dictionary where the key is the document name and the value is a list of all unique words in that document
adjusted_index	**build_adjusted_index(data)**	Build a dictionary where the key is a word in the corpus and the value is computed as logarithm in base 10 of (total number of documents in corpus / number of documents containing that word)

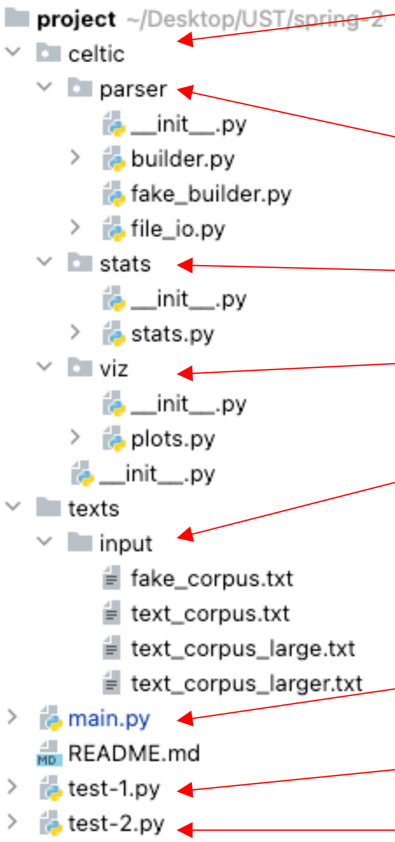
The stats module: this module defines functions that given certain parameters, compute statistics and return a value. Which in turn can be used elsewhere in your code. You have to define and implement these functions.

Function signature	Returns
`most_common_word(data)`	The number of times the most common word appears in the corpus, the parameter data is a container to be used for such task.
`get_largest_document(data, unique=False)`	The document with the highest number of words. If the second parameter is not given (the default) returns the document with the highest number of unique words. **data** is a container to be used for such task.
`avg_length(data, document)`	The average length of the words in a document. The parameter data is a container to be used for such task the parameter document is the name of the document (string).
`get_words_letter_end(data, letter)`	A list with all words in the corpus that end with a given letter. The parameter data is a container to be used for such task the parameter letter is a string, indicating the letter.
`get_words_letter_begin(data, letter)`	A list with all words in the corpus that begin with a given letter. The parameter data is a container to be used for such task the parameter letter is a string, indicating the letter.
`get_words_size(data, size)`	A list with all words in the corpus of a given size. The parameter data is a container to be used for such task the parameter size is an integer, indicating the size.
`compute_avg(data)`	Computes and returns the average size of words in the corpus. The parameter data is a container to be used for such task.
`get_words_grater_avg(data, avg_size)`	Returns a list with all words in the corpus whose size is greater than the corpus words average size.

The viz module: (This module is optional and if implemented will be a bonus on the grade) various functions that receive some arguments and plots bar chart graphs using the Turtle python package (there are tons of tutorials on-line about using Turtle for creating graphs). If plotting a bar chart involves too many bars, limit it to 15. You have to define and implement these functions.

1. plot_top_k_freq_words_doc(data, document, k), plots the counts of the top k words in one document.
2. plot_bottom_k_freq_wors_doc(data, document, k), plots the counts of the bottom k words in one document.
3. plot_freq_word_doc(data, doc_list, word): plots the counts of the “word” on the list of documents passed in doc_list.

Structure of the project. Below is the structure of directories and files of your project.



```
project ~/Desktop/UST/spring-2
  celtic
    parser
      __init__.py
      builder.py
      fake_builder.py
      file_io.py
    stats
      __init__.py
      stats.py
    viz
      __init__.py
      plots.py
    __init__.py
  texts
    input
      fake_corpus.txt
      text_corpus.txt
      text_corpus_large.txt
      text_corpus_larger.txt
  main.py
  README.md
  test-1.py
  test-2.py
```

celtic: directory containing the source code (python scripts) for your package. Inside this directory, there are other directories. Each directory is a module (with one or more python scripts), where functions are defined.

parser: module that defines functions for processing text and populating dictionaries

stats: module that defines functions for computing statistics

viz: module that defines functions for plotting bar graphs.

texts/input:
directory with input text file(s)

main.py this is the main script, I will use for grading, don't change anything in it

test-1.py this is a test script for one team member

test-2.py this is a test script for the other team member

Progress reports: To be submitted via Blackboard as a pdf document describing progress made during that week (each team member submits its own report) in pdf form (please no Word documents). Be very specific. Vague and generic statements are not allowed, for example: 'Everything is going fine' or 'We completed three functions', rather it should be something such as:

"We defined function cool_function(x,y,z) that computes statistic A, initially we struggled with getting the correct values, but finally realized that the variable was not properly referenced because x, y, z."

The report should include:

- Photograph(s) or scans of the work you have done in that week in paper (both team members have to include photographs).
- What has been done since the last report.
- What difficulties you are encountering.
- What strategies you will adopt to resolve the difficulties.
- What did you learn in the past week.
- Any issues coordinating and communicating with your teammate (e.g. teammate doesn't reply, make time to work or respond to requests).

Every Monday by 11:59PM, a push of whatever code you have so far has to be pushed to GitHub (you can have as many commits and push as you want; in fact, I strongly suggest to do so).

The do's:

1. Some initial code (aka glue code) for you to start the project, but read the description of the project, and bring questions to class.
2. Plan and organize the project, set concrete dates and times to meet on Zoom or in person, not just text each other.
3. Spend a good amount of time digging into the code, understanding how the modules are connected to each other, how things work, how variables and functions defined in another module are accessible in a different one.
4. Use the debugger and breakpoints to understand the flow of the code.
5. Coordinate/communicate with your teammate about the project.
6. **Start early to work on the project** and ask questions early in the process. If you get stuck in some part of the project ask about it, use the peer-mentor hours or my student hours (aka office hours) so help can be provided.
7. If you use code found on the Internet or other sources, please cite the source, including a comment on your code with the link to the source.
8. Both team members are equally responsible for the project.
9. Ask questions either in class or via GitHub discussion board about the project, don't be afraid of asking!

The dont's:

10. Do not wait until a week before the due date to start working on it.
11. Do not copy code from other teams (this has happened in the past and is cheating, not cool!). You can talk to other teams about issues or things you don't understand, or general ideas on how to approach something, or help about an error you can't figure out, that it's fine! But just copying and pasting or copying and changing some variables here and there is not allowed!
12. Do not just put code for the sake of pretending that work was done, the code has to run, and I spend a good amount of time grading the project!
13. Do not let a relative, friend, boyfriend, girlfriend, etc. write the code for you. It's is very easy for me to spot code not written by you. Of course, you can ask someone else for help and ideas on understanding why something is not working, how to fix a problem, but in the end, it should be your code!
14. Do not pay someone else to do the work for you. This is cheating!

Resources:

- About packages and modules in python: https://docs.python.org/3/tutorial/modules.html
- Reading and writing files: https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files
- Dictionaries: https://docs.python.org/3/tutorial/datastructures.html#dictionaries
- https://docs.python.org/3/library/stdtypes.html#mapping-types-dict
- Tuples: https://docs.python.org/3/library/stdtypes.html#tuples
- Lists: https://docs.python.org/3/library/stdtypes.html#lists