420-LCW Programming Techniques and Paradigms - Final Project

Instructor: Robert D. Vincent

Semester: Winter 2019

Basic requirements

Find a topic that interests you and write a program that addresses some aspect of that topic. **You may work in groups of 2 or 3 if desired.** You need not use specific techniques learned in the class, but all the better if you do. The main goal is to encourage you to explore some aspect of computing in a bit more depth.

While Python 3 is preferred, other languages are acceptable as long as it is freely available on Linux and/or Windows. Acceptable languages include Java, JavaScript, PHP, Perl, Ruby, Scala, C#, C++, and C.

Feel free to include libraries or tools as you wish, either those built into the language or those available on the Internet. You can also incorporate source code from public open-source projects, but you *must* document what tools, libraries, or sources you used, and you must have done some real work to adapt or extend any "found" code.

Try to make use of good software design principles, especially object-oriented design where appropriate. Think carefully about how to break your problem up into classes, functions, and/or files.

Topics

Here are a few suggestions for topics for your project. Students may pick anything from this list, or come up with your own idea. More than one student or group can choose the same topic as long as you do independent work. Discuss with me if you need more ideas.

1. Implement a class to represent a complex mathematical object, such as a matrix or polynomial. For this course I would expect the class to implement most of the applicable mathematical operators in Python, among others. Thorough testing and documentation will be crucial with this sort of project.

2. Create a program to solve a puzzle or solitaire game, such as Sudoku, a solitaire game, the 8-queens puzzle, "Grid Lock", etc.

   Consider whether you can solve the problem with the method used in assignment 2. For this you will need to invent a class that represents the "board" or the state of the game, and can simulate game play. Coming up with a metric for optimizing the search can be tricky, ask me if you need ideas.

   Other solution approaches are allowed. For very simple puzzles, consider making a visualization of the puzzle solution using `tkinter`.

3. Write a program that efficiently evaluates a board position for the game Boggle (https://en.wikipedia.org/wiki/Boggle) by finding all of the words that can be created with an arrangment of letters. This can be done with a variant of depth-first search. I can provide sample data and dictionaries for this.

4. Write a program to implement a simple two-player game like Othello or checkers. For two-player games your program should implement a computer player, but it can be very dumb (e.g. it could choose moves randomly).

   If you want more game ideas, take a look at this page: http://inventwithpython.com/blog/2012/02/20/i-need-practice-programming-49-ideas-for-game-clones-to-code/.

5. If you have a specific area of science or medicine that interests you, implement a program to solve a problem in that domain, such as writing a program to create or analyze a psychology experiment, simulate a physical system, or process any sort of scientific data. The project may just reproduce existing work.

6. Find a machine learning dataset from Kaggle or UCI and formulate your own machine learning model, using a free package such as `scikit-learn` or `TensorFlow`. If possible, compare your results to those previously published. Also consider implementing two different ML methods and comparing the results.

7. Pick a fairly advanced (junior 5 or senior 3-5) problem from the past Canadian Computing Competitions and provide your own working solution. Be sure to meet both the input/output *and timing* requirements.

8. Find a useful Python package and write a program to take advantage of it. Some possibilities:

   - `matplotlib` - 2D data visualization.
   - `scipy` - scientific and technical computing.
   - `scikit-learn` - machine learning.
   - `BeautifulSoup` - read data from web sites.
   - `socket` - low-level network communication.
   - `imageio` - read/write images (jpeg, png, etc.)

9. Find an interesting data structure or algorithm that we have not discussed in class, and write an implementation of it. This could include many possibilities:

   (a) Balanced trees like red-black trees or AVL trees.

   (b) Skip lists.

   (c) Radix sorting.

   (d) Compute the convex hull of a polygon.

   (e) Compute the Huffman tree of a text file (part of a compression algorithm).

   Ask me for details. For any of these examples, you'd have to provide both the implementation of the data structure or algorithm, probably as a class, plus a reasonable amount of code that tests the class.

10. Do some basic natural language or text processing, which could include simple full-text search program using an inverted index with some keyword weights based on document and word frequency. For example, you could implement a simple search process that would identify the document that best matches a search phrase. To make it interesting you should use a set of real documents, like some classic literature or public-domain news stories. I can help with ideas for appropriately-sized data sets.

What to submit (for all students)

1. The code for your project. This should be submitted electronically, but I would also like a paper copy if practical. Of course your code should be clearly structured and divided into separate files if appropriate. The rules for assignments still apply - comments and docstrings should be provided, as well as an identification section.

2. Since the program you'll be writing goes beyond the size of a simple course assignment, your normal in-line comments are not sufficient to document it properly. You must therefore write 3-5 double-spaced pages of external documentation to be submitted both electronically in Word or PDF format, and with a paper printout. This documentation has two sections:

   (a) A "mini-manual" for the user, explaining what the program does and how to use it, but not the internal workings.

   (b) A few paragraphs of a "design guide" for fellow programmers, describing the design and programming techniques used.

For epreuve synthese students only

1. The Abstract: this must fit on one page and explain the content of your project in simple terms. Do not go into technical details. At the end list all the keywords. Acceptance or rejection of the abstract will be determined by its clarity, quality of English, and presentation (e.g. no typos or spelling errors).

2. The government-mandated Self Assessment Grid. Answer as many questions as you can. Not all questions on the grid are relevant to your project but many will be. Read it closely, and either type or write legibly.

Presentation

Make it look professional: hand in the paperwork in a folder, clearly identifying the title, yourself as the author, and the course for which you are submitting it. Submit an electronic version as well, via Lea. Of course, keep a copy for yourself.

Deadlines

For the epreuve synthese, get me an abstract as soon as possible.

The rest of the project is due on the last day before General Education week.