420-SNT-MS Object-Oriented Programming – Final Project

Instructor: Robert D. Vincent

Semester: Fall 2025

1 Introduction

Choose a topic that interests you and write a program that addresses some aspect of that topic. **Work in groups of 2 or 3**. You should use specific techniques discussed in the course, especially classes and graphical interfaces. 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. Possible languages include Java, JavaScript, PHP, Go, Ruby, Rust, 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.

1.1 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.

Please send me your preferred topic by Mio before you start any serious work. I must approve your topic (and group arrangements) before you get started. Of course, we can negotiate changes in the scope of the project as needed, in case we discover something is more difficult than anticipated.

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.

   For this you will need to invent a class that represents the "board" or the state of the game, and can simulate game play. Optimizing the search can be tricky, ask me if you need ideas.

   If your choice is simple enough, try to make a visualization of the puzzle or game using `tkinter`.

3. 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 this player need not be very sophisticated. Your game should have some interesting or challenging feature - for example, a very nice graphical user interface, a complex set of rules, or a somewhat intelligent computer player.

   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/.

4. 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 be based on previously published work.

5. Find a useful Python package and write software 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.)

6. 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) Compute the Huffman tree of a text file (part of a compression algorithm).
   (d) Encryption or data compression algorithms.

   For any of these examples, you'd have to provide both the implementation of the data structure or algorithm using classes, plus a reasonable amount of code that tests the class. I'd also expect a brief but careful discussion of the performance characteristics of your code.

1.2 What to submit

1. The code for your project. This should be submitted electronically. Your code should be clearly structured and divided into separate files if appropriate. The rules for assignments still apply - comments and docstrings must be provided, as well as an identification section.

2. You must write three or more double-spaced pages of external documentation to be submitted electronically in Word or PDF format. This documentation has two sections:

   (a) A user guide, explaining what the program does and how to use it. This may include some basic information about the background of the problem solved, rules of the game, etc. Depending on your project, this may range from a couple of paragraphs to several pages.
   (b) A design guide for fellow programmers, describing the software organization and programming techniques used. This would be where you would mention what online sources you may have used, what tools or dependencies were required.

1.3 Marking

The project will be marked out of 30 points: 16 for code, 8 for documentation, and 6 for your presentation. Group projects will be assigned a single mark, which will apply to all members of the group.

The code will be evaluated for the following criteria:

- Organization (6 points) - Is the code clearly organized? Are modules, functions, and/or classes used to break the problem into meaningful and logical components? Are functions and variables given helpful names? Avoid including redundant, repetitive, or useless code, as this may affect your mark for this area.
- Correctness (6 points) - Does the program run as expected? Does it crash or contain serious bugs? Does it handle easily anticipated errors from the user?
- Sophistication (4 points) - Does the program solve an especially challenging or difficult problem? Does it use an advanced technique to improve performance? Is the user interface carefully thought-out?

The documentation will be evaluated for clarity and completeness.

Each group will do a 5-10 minute demonstration of their project, including a brief question-and-answer period.

2 Deadlines

The presentations will be done in class on November 24th and 26th.

The final project must be submitted on Omnivox by 5pm on November 27th.