DAT-119 – Python 1

**Final Project**

For your final project, I expect you to use many of the concepts we’ve learned in class. At a minimum, I would expect to see

1. some kind of repetition structures (loops)
2. some conditionals (if/elif/else)
3. multiple functions (including using main() properly, with no global statements or variables)
4. input to and/or output from files

Use of NumPy, Pandas, Matplotlib, CSV, or any other imported module is entirely optional, but may make your life easier, especially if you’re doing one of the data-centric projects.

I expect your program to be between 100 and 250 lines of code, just as a guideline--if it’s getting much longer than that, you are probably working too hard and might consider scaling back a bit. (Unless you’re having fun. I’m happy to grade a thousand-line program, if you’re up to writing it; I just don’t want you to bog yourself down unnecessarily.) If you’re using a data science module, you might end up a bit short on your line count; on the rubric, I refer to that as “sufficient complexity that a shorter program is reasonable,” and you’ll get full points, as long as you do a fairly detailed analysis of your data.

I don’t *expect* you to use concepts, tools, or modules we haven’t gone over in class, but you are now *allowed* to, if you want. Cite any external resources you use, of course.

Your project should be complete, production-ready, and uploaded to Blackboard (twice—once as an assignment submission and once in the Discussion Board) by midnight on Sunday, December 13: there should be good interaction with the user (helpful prompts, helpful error messages upon invalid input, graceful ways of exiting), and it should follow the style guide for our class, with readable, well-commented code.

Note: I’d like to be able to share your projects with future students, and you or others might want to build on each other’s work in Python 2 or other future endeavors. If you don’t want me to share your project, or you don’t mind my sharing it *as long as I remove your name*, please let me know.

The two default project options (data-centric!) are as follows:

1. Pull in a data file, and run a bunch of statistics on it. (Mean, median, mode, standard deviation, all of that. But you might also count unique items, if it’s that kind of data set. You might look at how items are clustered, or where your quartiles fall.) Output something to the console, so the user knows it’s working, but also put transformed data and/or useful information into a file for later use. Maybe make a nice chart with matplotlib, if you feel like it.

I am perfectly happy if you want to use one of the data sets you have used/are using in DAT-102, as long as you do all of the work for *this* class in Python. Plus, you can compare the results of your Pythonic number crunching to Excel’s, which might be instructive.

If you want to do this project, but you’re having trouble finding a data file to work with, I’ll set you up with my favorite one. :) Also, a note: some data files have a lot of extraneous information. It’s OK to clean up the data a little bit in Excel before you do anything in Python. We want our data projects to be reproducible, though, so be sure to cite your data source, and keep (and share!) notes about any rows or columns you’ve deleted, or any other transformations you’ve done in Excel.

Potential sources of data: <https://data.wprdc.org/dataset>, <https://data.pa.gov/>, <https://www.data.gov/>, <https://www.kaggle.com/datasets> (I listed the Western PA, PA, and USA data portals, but if there’s another location that’s more interesting to you, I bet they also have data sets available! Kaggle’s got many more general data sets, and you’re welcome to go out and find or make your own data to analyze.)

1. If text is more your thing than numbers, you can do an analysis of a large text file such as you’d get from Project Gutenberg. (I *strongly* recommend constraining yourself to a single short story or long poem, rather than a whole book.) Text is data. :)

You know how to count how often certain words appear, and that’s certainly fun, but I expect you to build on what we’ve already done; maybe you want to look at the average length of Charles Dickens’s sentences or the prevalence of semicolons versus commas, something like that. Maybe you want to count *every* unique word and output a ranked list of how often words appear. The sky is the limit, so if you find you have fewer than 100 lines of code, you probably need to do a bit more analysis on the file.

If you aren’t focused on data analytics, or if you’re feeling a little burned out on data stuff right now, you’re free to come up with your own project; we should talk it over ahead of time, so I have some idea of what you’re thinking of and can let you know if the scope seems right. You’re also free to choose a project from the menu below, which I have tried to rank “least challenging” to “most challenging”:

1. Let the user make a list of books or song tracks or something. Each individual list item should be thought of as a “record,” with multiple information items attached to it, such as Title, Author/Artist, Year, Genre, and maybe User Rating. The list should be persistent between sessions (meaning: it lives in a text file). The user should be able to Create, Read, Update, and Delete items (“CRUD” is a term that web developers use regularly), and the user should have the option to view items that fall into the same category; for instance, you might let them pull up all items in a certain genre, or all items by a certain artist. This is the easiest project option, because it builds off of the list app we’ve already made; however, it should still present some challenge: dealing with full records instead of single strings and letting the user pull up only a subset of the file will be non-trivial to implement.
2. A “book” of mad libs. (3-5 text files with words strategically removed, to be replaced by the user. Either the user can choose between the stories by title, with a menu, or you can present one of the files randomly to the user when the game starts.) Here is Wikipedia’s article about Mad Libs, so you can see some examples: <https://en.wikipedia.org/wiki/Mad_Libs> This project is less about the content of your mad libs (meaning: I don’t care if you just grab a couple of paragraphs from Wikipedia to use as your “stories,” as long as you cite them) and more about coming up with a generalizable way to read in formatted text, parse out what needs to be added to fill in the spaces, prompt the user appropriately, and display the output nicely. It would be nice to include documentation with your program that tells people how to generate their own Mad Libs, to be parsed by your program.
3. There are some games that, by the end of Python 1, you’ll be prepared to write. You could make a Sudoku solver, a smart Tic-Tac-Toe (first pass: the computer plays at random; improvement: it tries to win, after letting the player play first), 2048, or Battleship. Most of these don’t inherently require file read/write operations to work, but perhaps you could have a “high scorers” file, or keep track of how often your program beats the player. I ranked this option “most challenging” because we haven’t talked much about how you’d construct a grid system and because most games break 250 lines very quickly. If you’re motivated, though, making a game can be pretty fun!