SENG 265 Portfolio Term Project

Wesley Ducharme

Student ID: V00974267

Netlink ID: wducharme

Jupyter notebook

Jupyter notebook is an environment that gives users the ability to make documents with code, equations, narrative text and more. Its core functionality is to make a shareable document that has the ability to contain and combine code, plain text, data, and visualizations for data. Some of these visualizations include plots, graphs, charts, and 3d models with interactive controls.

Simple Jupyter Notebook markdown:

- Making headings: To make a heading you add a # to the front of the line you want to be a heading (ex. # Heading). The number of # symbols will determine the size of the text (heading level).
- Making paragraphs: Use a single blank line to separate one or more lines of text. Don't indent paragraphs in markdown.
- Line breaks: To create a new line, end the previous line with 2 or more spaces and then press return (enter).
- Bold text: To make a part of some text bold you do two sets of double asterisks. The area in between the 2 sets of double asterisks is the text that will be made bold.
- Italic: To make text italic you do 2 sets of one asterisk.
- Bold and Italic: Do two sets of three asterisks.
- Lists: A dash - at the front of a line followed by a space makes an unordered list. Add numbers instead to make an ordered list.
- Links: To make a link do [some link name] (some URL address). The closed brackets being what you want to name your link and the open brackets being the URL address of the link.
- Images: To add an image, add an exclamation mark (!), followed by alt text in brackets, and the path or URL to the image. For example, ![Some photo name] (some directory path or URL to the image).

More Notebook markdown and LaTeX markdown instructions for proper formula appearances here. This link will also provide proper examples of how to make text bold and italic.

LaTeX examples:

- To get a nice looking mathematical formula you do two dollar signs with the formula in between the dollar signs.
  - example: Euler's identity: $e^{i\pi} + 1 = 0$
- To make greek letters type \ then the letter name. Make the first letter in the name uppercase if you want the uppercase greek symbol.
  - example 1: \omega ω
  - example 2: \Omega Ω

Bash

Bash is a shell (or command/scripting language), for the GNU operating system that reads commands and runs other programs. Bash's main advantages are its high action to keystroke ratio, ability to automate repetitive tasks, capacity to access networked machines, and its large number of commands that do many specific tasks.

Command functions:

- Navigating files and their directories
- Working with files and their directories
- Pipes
- Filters
- Loops
- Shell scripts
- Search functions

Many of these commands are found in these links here and here where their function and many options are covered.

Bash also supports a text editor called Vim which allows the user to go into a file or program and edit text or code. Vim supports its own large number of commands for ease of use and increased efficiency. Many of those commands can be found here.

Bash also has commands such as chmod which determine permissions for who gets access to certain files and to what extent they can interact with those files. This link here covers in detail how the permissions work and how to set them using the chmod command.

C programming language

This course's first assignment is the first time I have worked with C. Now having done the assignment of making a Unix filter, I am familiar with some differences C has in comparison to languages I have worked with before like Python and Java. Some of these differences that stood out the most to me when working on the filter were how C manages strings, and how in C you have to manually manage the memory allocation. Other big differences I noticed and got familiar with was C's support of pointers for direct memory manipulation as well as setting up structure types. I am excited to work with C more throughout seng265 since it was the top programming language for a long time and still is a very prevalent language today.

Python Programming language

Python was the first language I learned in the course csc110 here at Uvic. I learned the basics of programming in python from simple variable naming and math operations to lists, file I/O, user input, and for/while loops. I am also familiar with the making of dictionaries, tuples, and object classes from my experience in csc110. I look forward to working with python more in seng265 as it will be good to review and brush up on the skills I have already learned in previous courses.

Git

Git is a version control system which allows many people to collaborate on projects. When accessing Git you are accessing a remote repository hosted on services such as Github or Gitlab. Allowing you to pull files to your local machine, make changes, then push those changes back to the remote repository. It allows a lone developer or a group to easily keep track of changes that were made in case of future problems and more efficient collaboration through documentation.

The Git workflow is managed through roughly this sequence of commands:

1. git clone: Creates a copy of the remote repository in a specified folder of your local machine.
    - Command: git clone 'url' 'localFolder(option)'
2. git status: Lists updates and changes you have done to your copied repository
    - Command: git status
3. git add: Adds a change of a file in the local repository to the index (staging area).
    - Command: git add 'file'
4. git commit: Creates a record of the staged changes in the local repository
    - Command: git commit -m '"text message"'
5. git reset: Undoes the changes done in git add or git commit
    - Commands:
        - git reset --'filename' (resets git add)
        - git reset --softHead~1 (resets git commit)
6. git push: Uploads the local repository changes to the remote repository
    - Command: git push
7. git pull: Incorporates changes from a remote repository into the current local repository.
    - Command: git pull

Weekly recap in seng 265 Part 1

Week 1:

Week one we were introduced to a lot with the major things being Jupyter Notebooks and some history on operating systems. Some of this history included the where and when Unix, Windows and Mac operating systems were developed as well as their different versions. I was pleased to see we will be doing work with Jupyter Notebooks as I am interested in data science and getting familiar with this tool will be very valuable. We also went over what we will be covering in seng265 and I am very excited to learn C and do those assignments as I have only worked with Python and Java in previous courses.

Week 2:

Week two was all about Bash (Bourne again shell) and learning its many specialized commands like grep, head, tail and touch, chmod and others. This week we were also introduced to the course reference platform. This was my first time working on a server. I thought it was really interesting how it was all set up. I plan on working on assignment 1 directly on the reference platform using vim as it will be the easiest way to ensure everything works properly when it is being graded and I want to practice using vim and navigating the reference platform.

Week 3:

Week three we covered a lot of bash commands in depth like grep, head, tail and touch, chmod and others. This week I also started to go into depth with the Bill Bird c videos for assignment one. I am mostly going to be comparing my experience with C to Java as I am most familiar with Java. So far C doesn't seem too different from Java except for some key things I am noticing. Like how strings are worked with as well as memory management and C's support of pointers.

Week 4:

For this week I was reviewing the Bash commands and their functions as well as reviewing the many examples of those functions given to us in class. I overall felt pretty comfortable with the material for midterm 1 and think I am getting to a good point with the C videos that I know enough to start on assignment 1. I have the design and structure of the program all sorted out and know roughly what functions are going to need to be added and how they connect to each other. I am still trying to figure out how to exactly implement all of it in C though.

Week 5: Finishing assignment 1 and learning Git

This week I finished assignment one and thanks to Felipe's presentation on git and the lab exercise I was able to correctly submit assignment 1. These presentations helped me alot when it came to understanding how git works and its purpose in industry. The lectures covering how C pointers work helped clarify that topic for me as I was having a bit of trouble fully understanding what exactly they did and how they worked. Overall the C content this week helped clarify some things that I was not too sure about from just watching Bill Bird's C videos.

Using chat gpt as a learning tool Part 1

Overall I would say my experience with chat gpt was good. Just asking it to make code for you doesn't completely work as it will be lacking a lot of other aspects of the program you are making and its purpose. It is however great for getting a clear explanation on definitions and basic functions. It is also great for visualizing how things might need to be structured as it can readily provide examples of say for example a function's structure and how it might interact with other functions in a program. I believe its ability to provide examples with clear explanations is its greatest strength as a learning tool.

Python Libraries

Pandas:

Pandas is a powerful tool used for data analysis and data manipulation when working in python. It's usefulness comes in its ability to make data structures and do work on those data structures. This work includes sorting, editing, and organizing data. It provides two types of classes (data structures) for working with data.

These classes are defined as:

1. Series: A one dimensional array holding data of any type.
2. DataFrame: A two dimensional data structure that holds data such as two dimensional arrays or tables containing rows and columns. Similar to a spreadsheet or dict of Series objects.

Basic methods

- make Series: s = pd.Series(data, index=index) (Note: data can be a lot of types. Ex: dicts, scalar values, strings)
- make DataFrame: df = pd.DataFrame(data)
- read csv file to DataFrame: df = pd.read_csv(file)
- drop columns from existing DataFrame: df_drop = df.drop(["some_column1", "some_column2"])
- sort with respect to some value: df_sort = df.sort_values(by=value, ascending=True/False)
- get the first n rows: df.head(n) (Note: default n is 5)
- filtering rows: results_df = results_df[results_df['positionOrder'] == 1]
- merging DataFrames: merged_df = results_df.merge(drivers_df, on='driverId', how='left')

Pandas website link for more details and functionality.

MatPlotLib:

Library used for making visualizations. It graphs data on figures which can have one or more Axes, and plots points on x-y, z-y-z, theta-r and more.

A Figure keeps track of all the components that make up the visual. These components are called Axes which are attached to the figure. These Axes include one to three Axis objects. Axes are the main entry point to work on an interface and have the most plot functionalities. The Axis object sets the scale and can generate labels or marks on itself called ticks. Lastly, everything that you can see on a figure is called an Artist. These Artists are what is drawn on the canvas when the Figure being made gets rendered.

Matplotlib website link for more details and functionality.

NumPy:

This library is mainly used for scientific computing. Providing an object called the ndarray which is a n-dimensional array of homogeneous types. The ndarray is different from regular arrays in that they have a fixed size and require less code to work on them. Most of the operations done on ndarrays in NumPy are done with pre-compiled C code. Making the operations execute fast with little code needed. For example if were to multiply all the elements in an integer array A by the elements in integer array B and get a new array, it would only take the single line of code "C = A * B'' where C is the array holding the new values. NumPy is easy to read and able to operate fast due to its vectorization of its code.

NumPy website link for more details and functionality.

SeaBorn:

A library based on MatPlotLib that is used for data visualization. Giving very detailed statistical graphics. It works very well in tandem with pandas dataframes and sequences. The plot functions that are in the library work on arrays and dataframes. There are many examples of what can be done with this library on their website and these include scatterplots, histograms, boxplots, modeling linear regression and many more. Many of these visualizations give the ability to see things like data distributions, statistical relationships, statistical estimations, and many more data characteristics.

SeaBorn website link for more details and functionality.

SciPy:

This library is builds on what Numpy can do and is mainly used in scientific computing. It provides a wide range of useful math and statistical functions. For statistics it provides functions for probability distributions, hypothesis testing, probability density functions, cumulative distribution functions and many more statistics functions.

It has a lot of math functions but some of the more notable math functionalities would be:

- Integration functions ranging from single,double,triple integrals, as well as integration for partial and ordinary differential equations.
- Optimization functions finding maximums and minimums, etc.
- More Linear algebra functions when compared to NumPy like eigenvalues, eigenvectors and better optimization for sparse matrices.

It also works well with other libraries used for scientific computing like MatPlotLib and Pandas.

SciPy website link for more details and functionality.

PEP 8

PEP 8 is a style guide for coding in python. The main purpose of the guide and why it should be followed as best as is possible is to write clear and easily readable code. This guide offers a way for there to be a similar and consistent style across many projects. This allows reading the code from one project to the next to be more efficient and understandable.

Below will be key aspects of this style guide in point form.

Code layout:

- For indentation 4 spaces per indentation level is the norm.
- Aim for a maximum of 79 characters per line.
- Match operators with their operands when spreading these operations across multiple lines
- Use 2 blank lines to surround top level functions and class definitions. Use 1 blank line to surround methods in a class. The use of blank lines can be used elsewhere in functions when separating logical sections.
- Use UTF-8 for code in the course python distribution.
- Put imports at the top of the file after comments and docstrings. Imports should be done on separate lines.

Comments:

- Inline comments are done with a single hashtag and space (# comments)
- Multi-line documentation strings should be written after the def line of a function and start and end with triple quotes on different lines.
- Single-line docstring with start and end with triple quotes on the same line.

Naming conventions:

- Names that the user can see should be based on the usage instead of implementation.
- Some common naming styles are single lowercase letter, single uppercase letter, lowercase, lower_case_with_underscores, UPPERCASE, mixedCase.
- Modules and packages should all have lowercase names.
- CapWords naming convention should be used for class names, type variable names, and exceptions.
- Lowercase and lowercase_with_underscores should be used for variable, global variable, instance variables, method names, andfunction names.
- Use “self” for the first argument in instance methods and “cls” for the first argument in class methods.
- Type hints should be done as follows: (num: int) and/or (num: int = 12)

Pythonic Programming and Comprehensions

Pythonic programming is a code writing style that best adheres to not just syntactically correct python code but also code that follows the standards of the python community. These standards include code being as readable, simple, and clear as possible. One form of code that are considered to be pythonic are comprehensions. Comprehensions are a more concise and readable way to create lists, sets, and dictionaries. However they may lose readability the more complex the logic for a problem might become and can also become harder to debug as a result of this.

The syntax of comprehensions and some examples:

- new_list = [expression for item in iterable if condition]
  - example 1: cubes: list = [x**3 for x in range(10)] makes a list of cubes from 0 to 9.
  - example 2: odds: list = [x for x in range(21) if x % 2 != 0] makes a list of odd numbers from 0 to 20.
- new_set = {expression for item in iterable if condition}
  - example 1: letters: set = {ch for ch in word} makes a set of unique letters form a word.
  - example 2: primes: set = {x for x in range(5, 31) if all(x % y != 0 for y in range(5, x))} makes a set of prime numbers from 5 to 30.
- new_dict = {key_expression: value_expression for item in iterable if condition}
  - example: cubes: dict = {x: x**3 for x in range(10)} makes a dictionary mapping numbers to their squares.

Another form of code that is considered pythonic is type hinting. Type hints allow you to make the data type of your variables, arguments, etc, clear and by extension making the code more readable.

- examples:
  - (number: int = 1): specifies number is an integer type.
  - (decimal_number: float = 1.0011): specifies decmial_number is a float type.
  - (word: str = "hello"): specifies word is a string type.
  - (square_dict: Dict[int, int] = {4: 16}): specifies square_dict is a dictionary of integer types.

One last example of code that is pythonic is slicing. Slicing allows you to extract a section from a sequence that supports slicing. These sequences can include lists, tuples, strings, and others. Slicing is considered to be pythonic as it allows you to set a clear range of how much of a given sequence you want to extract.

- examples of slicing given a list of integers list = [2,4,5,6,8,4].
  - list[2,5] extracts the elements from index 2 to 4.
  - list[:4] extracts the elements from the first index to the third.
  - list[3:] extracts the elements from index 3 to the last index.
  - list[::3] extracts every third element.

Implicit and Tacit knowledge

Demonstrating implicit and/or tacit knowledge is very important when working for a client to design or make a product. This is because you can never fully rely on the client to understand the full extent of what kind of work may need to be done to reach their desired outcome/finished product. That is our job to see where problems might arise in development and ask questions to clarify specifics from what may be a generally ambiguous instruction or feature they want done. It also may be that the client is very aware of all the work and specifics that need to be done but doesn't say it explicitly as they see it as too small a detail or as a known assumption. It is very important to recognize these instances to ensure that things can be planned as best they can and so you can minimize changes/bugs that may show up down the line in development.

Assignment challenges

Assignment 2

In assignment 2 I wouldn't say that it was a super challenging assignment for myself as I have used python before in csc 110 but I did find it interesting and enjoyed doing it. This is because I used Pandas extensively when implementing my solution for the assignment. It is my first project using a python library to do most of the work and it really showed me how useful these libraries can be. Pandas made it simple to filter, order, and manage the data that we were working on and did this with minimal lines of code that was easy to read. Seeing how useful Pandas is, it encouraged me to explore deeper into the library and other libraries that work with data like Seaborn and Numpy and how they give more options and make things simpler when working with data.

Assignment 3

I have found assignment 3 to be the most challenging so far. Designing how the program will be structured and what functions I will need to do certain tasks wasn't super challenging. That part was very similar to assignment 1 and I have used nodes and linked lists before in Java and Python. What I found challenging was the dynamic memory allocation. Working with dynamic memory allocation for the first time on a project left me dealing with a lot of segmentation faults as I was not 100% sure where to free up memory and had to deal with some small errors like null pointers that I had lost track of on my first round of implementation. I found and fixed all the problems though and I feel I learned alot about how C memory allocation works (specifically pointers and dynamic memory) having dealt with these challenges.

Weekly Recap Part 2

Week 6:

Week 6 has been all about Python. We covered many of the basics of python like functions, type hints and immutable types. I have decent familiarity with Python since taking csc 110 so I understood a lot of these topics well already. The new things I learned though were how python's memory allocation works and how it is all handled by the interpreter with objects going on the heap and variables pointing to objects going in the stack. Another new thing that we didn't cover extensively in csc 110 would be the built in functions. We covered a few like range(), sum(), and abs(), but did not use any of the seemingly more useful ones like sorted(), repr(), and enumerate().

Week 7:

More Python information was covered this week where we talked more about mutable(lists, dicts, sets, etc) and immutable types(ints, floats, strings, tuples, etc) and how mutable types are objects that can be changed after they are made, where immutable types are objects that once are made cannot be changed. If you would want to change an immutable type variable you would be making a new object. Another big topic this week was collections like tuples, lists, dictionaries, and sets. These collections I have used a lot in and since csc 110 so nothing talked about here was new to me but it was still a good review. What was new to me this week was the talk of Pythonic programming/ Pythonic code. Namely some examples of this kind of code were the comprehensions for lists, sets, and dictionaries and their syntax. I thought these comprehensions were great demonstrations of how more pythonic code can have its benefits for readability as well as often being a lot quicker to write.

Week 8:

This week we covered both some C and Python. For python we were shown some slicing for strings and dictionaries which I found very helpful. Though I learned about it in csc 110 I have hardly used slicing in any projects I have done in Python so this was a great review for me. I also found the demonstrations on more pythonic methods of file I/O very useful as I used them in assignment 2. The second part of this week we were back in C and specifically talking about how to allocate and deallocate dynamic memory. Though I can't say I fully understand how to properly integrate dynamic memory yet I now have a decent understanding of the function malloc, calloc, realloc, and free. We also talked extensively about structures but I had already researched and used structs a lot in assignment 1 so this section was more like a review for me.

Week 9:

This week I found what we covered to be interesting as we talked alot about interview questions and specifically Tacit/Implicit knowledge and its importance. It became clear to me very quickly the importance of this concept as it's mainly centered around clarity and communicating your understanding of requirements for a project/product when working with others/clients. Being able to notice and talk about hidden/ambiguous details that can possibly have a large impact on the workflow for a project can lead to greater efficiency and less changes down the line. We also talked alot about nodes and linked lists this week which is a concept I am very familiar with from csc 115 having worked with such data types and structures extensively in Java. However these lectures were very useful in determining the syntax to make these data structures in C and giving a review of the concepts.

Week 10:

Week 10 has mostly been about Python and specifically object oriented design in python. I worked with OOD alot in csc 115 but this is a great refresher as well as showing the best practices and syntax for making object classes and their instance methods. The many examples demonstrated in class will be extremely useful for the last assignment which is all about classes and objects. We finished the week off with SVG (scalable vector graphics) and some good examples of html code that we will be using for assignment 4.

Seng 265

Seng 265 is definitely one of the more useful second year courses for building a resume and getting ready for job applications. It has introduced a lot of new concepts around not just programming in C and Python but also important information about software engineering. Tools like Bash, Jupyter Notebooks and Git were all covered extensively in this course and are all very useful tools for a variety of jobs. The assignments also have acted as great projects to improve and demonstrate the level of understanding I have of Python and C. The skills I have learned from this course that I will definitely mention in my resume will be the further development of my C and Python skills, my usage of Bash and Vim while working in the lab server as well as my understanding of unix, and pipes and filters.

AI as a learning tool part 2

I have used chatgpt in pretty much the exact same way as I had in the first part of the course. That was a great way to look up basic and key aspects of concepts that we were learning in this course as well as using it to generate demonstrations and examples of these concepts. It is a powerful tool that in practice can do alot for you but if you are but you won't be getting a whole lot out of it if you are just copy and pasting what it outputs. It is often lacking an understanding of context and as a result can output information that may be partially right but may be missing things or even just be wrong. This is why it is important to practice some critical thinking when using tools like A.I and to carefully read through its output as you will never learn anything by just copy and pasting.

Bibliography

[1] K. A. Aziz, “Learn how to write Markdown & Latex in the Jupyter Notebook,” Medium, https://towardsdatascience.com/write-markdown-latex-in-the-jupyter-notebook-10985edb91fd
[2] “Code style - the hitchhiker’s guide to python,” Code Style - The Hitchhiker’s Guide to Python, https://docs.python-guide.org/writing/style/
[3] “Pandas,” pandas, https://pandas.pydata.org/
[4] “Matplotlib 3.8.3 documentation#,” Matplotlib documentation - Matplotlib 3.8.3 documentation, https://matplotlib.org/stable/index.html#learn
[5] “NumPy documentation#,” NumPy documentation - NumPy v2.1.dev0 Manual, https://numpy.org/devdocs/index.html
[6] “Statistical Data Visualization#,” seaborn, https://seaborn.pydata.org/index.html
[7] “Scipy documentation#,” SciPy documentation - SciPy v1.12.0 Manual, https://docs.scipy.org/doc/scipy/