Term Portfolio - SENG 265

Isaiah Doyle, Spring 2022

Part 1

1. What is my background in Jupyter? What is the core functionality of Jupyter notebooks?

> Throughout my time programming and working with code, Jupyter has been relatively out of the scope of my projects. Jupyter notebooks are a tool that can massively increase integrity and help legitimize the work done by those in computer science or software development. For example, if one were to come up with some idea at some point while coding, keeping a journal would be a game changer when trying to prove that said idea was actually yours.

2. Summarize simple Jupyter notebook markdown including how to insert links and images.

> Jupyter uses a specialized markdown that allows for simple text styling for documents. Some examples of the syntax used by Jupyter are:
>
> - *Italics*: `*italics*`
> - **Bold**: `**bold**`
> - Insert link: `[link alias](url)`
> - Insert image: `![alt text](image url)`
> - Heading: `#, ##, ... ######`
> - Blockquote: `> blockquote`
> - Ordered List: `1.` , `2.` , etc.
> - Unordered List: `-` or `*`

3. How do you typset mathematical formulas including Greek letters using LaTeX in Jupyter?

With a general understanding of LaTeX, the hard part about inserting equations and special characters in Jupyter is over. To insert an inline equation, simply wrap the equation (of LaTeX sytnax) in dollar signs (i.e., `$ [equation $`). For block equations, wrap the equation with two dollar signs (i.e., `$$ [equation] $$`). For example, consider the definition of the continuous-time Fourier transform:

$$\mathcal{F}x(\omega) = X(\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t}\ dt.$$

4. What is my background in Unix/Linux? How prevalent are such operating systems in software development?

In the late '60s and early '70s, a group of researchers at AT&T Bell Labs began developing an operating system on a microcomputer called the PDP-7. The computer would quickly go out of style, but the operating system, now called Unix, was rewritten in C such that it was able to switch to new hardware as it came. Unix spread like wildfire among technology corporations and many different versions were created and are now used today (e.g., macOS).

Eventually in 1991, under the Free Software Foundation, the Linux kernel was brought to the table, and combined with the rest of the GNU project (i.e., a functionality that wrapped up the OS with a heavily functional bow), served as a totally free alternative to Unix.

Although I haven't diven into the structure of operating systems to much extent, I've been deeply interested in beginning to learn for a while. Having worked in teams on small and large development projects, it's become increasingly clear that Unix and Linux are huge in the tech industry and crucial to understand in order to pick up increasingly complex concepts in computer science. I can't wait to learn more! [1]

5. What is the core functionality of the Bash command and scripting language?

The Bourne Again Shell, or Bash, is the shell made for the GNU project, supported by the Free Software Foundation. Bash was intended to be a free, open source mix of C shell and Korn shell, but most importantly, a free alternative to Unix's Bourne Shell.

The purpose of Bash, and shells as a whole, is to translate commands between the user and computer to execute tasks effectively through a command line interface (CLI). This allows for a more intimate and powerful experience when working with your machine, and is imperative to an efficient workflow. [2]

6. Why is version control and configuration management critical in software development projects?

For large-scale projects, the development process is riddled with potential errors and without a solidified management plan in place, a project of any size can quickly become a total nightmare to work with. For this reason, version control and management technology is imperative in leading a successful workflow.

Version control allows teams and individuals to keep track of the development process in a way that makes room for error, and allows developers to take any number of steps back - or forward - when the need arises.

7. What are the core functionalities of Git, Github, and Gitlab?

Git is an open source version control and project management software that allows developers to organize their workflow as described in prompt 6. Github and Gitlab are external software that both allow users to make public and private repositories to allow other developers to contribute on whatever projects are pushed to Github/Gitlab via Git. The core functionality of Github and Gitlab are collaboration, which is absolutely vital for many large-scale projects in which sometimes hundreds of developers can be working on the same project at once.

8. What is my background in C programming? How prevalent is C in software development?

> I've been interested in learning C/C#/C++ for a significant amount of time, but due to either lack of motivation or time I haven't studied enough to merit any credit. Although old, C is present in almost all technology that surrounds us. Operating systems, web browsers, large programs, and small programs all rely on C in some way. Due to this, I'm super pumped for an excuse to actually focus on C programming.

9. What is my background in Python? How prevalent is Python in software development?

> I was introduced to programming through Python in late high school, and continued my studies in my first year of university in CSC 110. Since then, I've grown to love Java, React, and everything in between. I look forward to moving back into Python due to its intuitive design.

10. What are my personal insights, "aha" moments, or epiphanies I experienced in the first part of SENG 265?

> I'm no stranger to my terminal, but I had never used it previously for much more than moving about directories and making minor modifications to files. Now, with Vim in the picture as well, I am almost overwhelmed with the amount of potential bundled into nothing but a plain text command line.

Part 2

1. How can Git be used effectively in a highly distributed software development environment that involves multiple countries?

Today, Git is a behemoth in the tech industry, and for good reason. Through Git and some hosting service such as Github or Gitlab, as developers we are able to share project files in an organized way that allows not only for detailed version control capability, but the ability to collaborate on said files via the internet from any distance. This is huge, as large companies with employees over multiple countries are able to upload and download updated files in seconds. That is, all it takes to move changes from someone's computer in Indonesia to another's in Canada is a simple `git push` one one end and a `git pull` on the other.

2. Briefly describe the history of version control systems.

The history of version control can be largely divided into three generations, each relying and improving upon the previous generation.

First generation:

Long ago on Unix computers, it was expected that when working on a collaborative project, each team member would have a respective account on the same Unix machine on which they could make changes to the project one at a time. This led to the creation of the Source Code Control System (SCCS) by Mark Rochkind (Bell Labs) in 1972. SCCS was a way to store changes to files in a dedicated "s-file" for each file changed. To prevent necessary modifications from being overwritten by someone else, SCCS prevented others from making changes to the same file at the same time.

In 1982, Walter Tichy wrote the Revision Control System (RCS), which was a free alternative to SCCS and works backwards (i.e., checks out older versions by reversing changes) whereas SCCS applied a forward-change scheme

Second generation (centralized control systems):

A major downside of the first generation's version control systems is the inability to work remotely within a team. That is, both SCCS and RCS relied on each developer to work on the same Unix host in order to make changes to a project. Dick Grune wrote the Concurrent Versions System (CVS) in 1986 to combat this issue. The architecture of CVS utilized the foundation and format of RCS, but changes are saved to a remote repository on a server set up to host the project. Due to this, multiple users from anywhere with an internet connection were able to make changes to a project at the same time - as long as other users make sure their work is up to date before committing their changes.

Something that CVS failed to consider, however, is that if a commit were to fail (e.g., due to an unreliable internet connection), there was a notable risk of the file being corrupted somewhere along the process. Additionally, CVS only allowed users to commit one file at a time. In 2000, the team at Collabnet Inc. wrote Subversion (SVN) to solve these problems. That is, Subversion uses a similar centralized repository approach, but introduced features that were desirable over the relatively unintuitive CVS.

Third generation (distributed control systems):

In 2005, the great Linus Torvalds created Git which quickly became a timeless technology we still use today. The major difference between distributed and centralized control systems is the ability to work without a centralized repository and not have to rely on an internet connection to make changes to a project. With Git, each user is able to copy a larger repository to their machine and work offline until changes are ready to be pushed to said remote repository. This allows users to create branches and share changes within a team without the need to merge with the main repository directly.

Matt Mackall created another VCS in 2005 called Mercurial which was in the running with Git to be the dedicated system for Linux. Mercurial differed largely from other version control systems as it was the first to be written in Python (rather than C, as those previously listed were). From a distance, though, Git and Mercurial shared a similar goal.

As previously stated, Git seems to have "won" the race of version control systems yet - but as we're still on the forefront of an increasingly digital age, it's entirely possible for some other system to pave the way for some fourth generation! [3]

3. What are the most popular programming languages and why?

There are a lot of programming languages out there. It would take far too long to explain the history and purpose behind even the top 10 programming langauges today, but Below are a few of the more prevalent languages we use today:

C/C#/C++:

C, developed in the early '70s by Dennis Ritchie at Bell Labs, was - and still is - a behemoth in the industry. Due to the low-level control over memory C allows developers to harness, it became widely recognized for use of developing operating systems and crucial tools such as version control systems, compilers, and even other languages (e.g., MATLAB). Additionally, C was developed with the intent of ease of use across all platforms, and as such renders the language intuitive for cross-platform development.

C# and C++ share a similar structure to C, as C# was written to be a similar alternative to C, and C++ was intended to be an expansion. C# was written in 2000 by Anders Hejlsberg at Microsoft, with a goal to serve as an object-oriented alternative to C. C# is most often used for web and game development. C++ on the other hand, was intended to be an extension of C, with a host of new features - most importantly, similar to C#, object orienatation. C++ is at the forefront of programming languages today, with use cases ranging from game development to machine learning.

Java:

Java was designed by James Gosling in 1995 and was largely a higher-level alternative to C. Java is praised for its intuitive design (at least, by me!) and ease of use. Two major avenuse Java is most commonly used for is web (not JavaScript!) and mobile development, due largely in part to its portability.

Python:

In the '80s, Guido van Rossum designed the Python language with a dream of intuitive design and readability in mind. Rossum accomplished just that, with the high-level but powerful language we know today as Python. Python happened to be the first real programming language I learned, and I'm eternally grateful that I started on a language that allowed me to grasp the rudiments of programming without the unnecessary boilerplate and complex syntax as other languages such as C or Java. Even though Python is easy to grasp, it's an extremely powerful tool and is often overlooked. [4][5]

4. How important is learning C and Python to your career development?

Both C and Python have a huge range of possibilities, and although they share many possible use cases they both have differences that give them unique advantages in different fields. For example, when working with low level applications that require immense speed, one might choose to use C - whereas if making some kind of backend API where readability is important, one might use Python. This ranges widely in all fields, and thus there is no objective hierarchy in terms of which language is 'best'.

Due to this, having at minimum a general understanding of both C and Python nearly doubles the number of potentital job opportunities as employers may require an understanding of one or the other. Additionally, being exposed to both C (a functional, statically-typed compiled language) and Python (an object oriented, dynamically-typed, interpreted language) provides a much broader range of understanding in programming as a general practice. C and Python vary greatly in terms of functionality and understanding what makes each language tick is a huge step in learning computer science and software engineering as a whole.

5. Describe the fundamental differences between C and Python.

To elaborate further, C is much lower level than Python. That is to say, Python is known for its readability and intuitive design, but C is harder to get a hold of for good reason. C provides functionality far greater than Python when memory usage and runtime is of heavy concern.

Since C is statically typed (i.e., post-initialization a variable cannot change its type), it's much easier to keep track of every byte of memory and consequently guarantee that a program is using the minimum required amount of space to perform operations. Python, on the other hand, is dynamically typed (i.e., variable types and sizes can change at any point); therefore, memory usage and scalability can quickly become a problem.

Another big difference between C and Python that might not have a direct correlation to runtime or memory usage, is C's reliance on pointers. Python variables are relatively straightforward, in the sense that any variable can be passed into any function, modified, returned, and so on. In C, variables are most often pointed to directly in memory, rather than simply passed around functions. This allows the user more control over what may be happening to a specific area in memory, but in comparison to Python it can be a somewhat difficult concept to grasp and even be a source of confusion for those familiar with C's syntax.

6. How challenging was assignment 1 for you?

Assignment 1 turned out to be a huge commitment, and one that I became increasingly proud of as I spent more time polishing my solution. I'm not sure I would say the assignment was difficult, but having never worked in C much before it became an exercise of learning the syntax and functionality of C as soon as possible. Having ample experience sifting through documentation and Google search results, this task came fairly smoothly, and I'm proud to say that although the assignment took a ton of time, I'm certain that my code is both readable and efficient - at least enough for my first C program!

7. Describe your continued learning experience in SENG 265.

As mentioned in question 6, the first assignment of the semester really threw me head first into the world of C, largely unprepared. As soon as work in C began, I fell into a cycle of perpertual learning, and even working in Python today I'm still learning more and more each week. Additionally, although I was familiar with my BASH terminal and general Unix/Linux commands, I feel as though throughout the semester I've gained a much higher level of understanding of my terminal and I've seen my efficiency increase drastically in my workflow. Even though we may be studying relatively specfific concepts, most of what we're learning applies to more than meets the eye; thus, I'm looking forward to what we do next, and what else I may be missing out on!

Part 3

1. How useful is SENG 265 for building your resume for future co-op and job applications?

SENG 265 has been an extremely interesting experience as the course has took numerous turns through different technologies. The four major assignments in particualr have been a super interesting accomplishment to add to my Github profile and actually showcase the work I've spent so much time on throughout the semester. An empty Github profile can be a negative sign for potential employers, and even filling the little green commit boxes on one's profile can make a big difference on the surface level (i.e., the "roadside appeal," per se).

Additionally, this very Jupyter Notebook is a valid documentation of my learning and growth throughout the semester, and demonstrates in detail all that I've explored through my studies.

2. Which skills explored in SENG 265 are most valuable to you and your resume?

First and foremost, C and Python are total behemoths in the tech industry. The two are used for a vast majority of all software development, and having grown confident in my understanding of both languages is an imperative step in professional development for any aspiring developer.

Alongside langauges, Linux/Unix are also extremely prevalent in the industry. Having been a regular visitor of the co-op job board this summer, I've seen first hand how much employers value experience in and understanding of Linux environments. Coming into SENG 265, I knew rudimentary Linux commands, but I had no idea what Linux was in a more abstract sense. This course provided me with a much deeper comprehension of Linux concepts that i can confidently add to my resume!

Testing/debugging, pipes/filters, HTML/SVG, data structures, file I/O, and all that we learned this semester are super valuable for future development endeavors. Most importantly, however, SENG 265 has equipped me with a more efficient strategy when it comes to diving into complex problems or technologies. I feel that given the experience I've had in SENG 265 being thrown into assignments in which I've had little background, I have developed a more open-minded point of view when it coems to new concepts.

3. What is the difference between object based and object oriented programming?

Object oriented programming is known for its four fundamental features: abstraction, encapsulation, inheritance, and polymorphism.

Abstraction: the process of limiting details from the user's point of view by designing methods and variables such that only important information is presented to the user.

Encapsulation: the process of setting certain class variables to be linked to a class, such that internal methods can rely largely on the data already stored within an object, rather than requiring external input.

Inheritance: allowing classes to act as more specific versions of another pre-existing class, inheriting most data such as class variables and methods. For example, a `Vehicle` class might have class variables such as `max_speed` and `type_of_fuel` with methods `start()` and `stop()`. A `Truck` class would inherit the variables and methods, and might add some more specific methods like `turn_on_highbeams_to_blind_other_drivers()`. A `Plane` class, on the other hand, might add additional variables to the `Vehicle` class like `cur_altitude`.

Polymorphism: the process of designing classes such that the specific nature of methods and data might vary depending on the individual instance or sub-class. For example, an `Animal` class might have a `breathe()` method. For some animal classes such as `Fox` and `Beaver`, the inherited `breathe()` method will rely on the lungs and air. For a `Fish` class, on the other hand, the inherited `breathe()` method would rely on the gills and water.

Object based programming is similar to object oriented programming, but only allows for abstraction and encapsulation. That is, objects are often used and built in, but classes aren't used for constructing said objects and thus inheritance and polymporphism aren't applicable.

4. Describe the notion and motivation for typing and type hints in Python.

When given access to a pre-existing codebase, the initial step of digesting the intent behind someone else's code can be difficult. Python can be particularly difficult since it's a dynamically typed language, and any variable can be quite literally any kind of data without the type being specified. Typing and type hints is a way to remove much of the confusion caused by dynamic typing by specifying the types of variables in a program. Consequently, this leads to fewer mental obstructions when reading code, and fewer potential bugs when passing data along.

Code readability is one of the most important aspects of programming, second only to the code actually running. Thus, typing is a crucial step in writing quality code.

5. What are the big advantages of Python classes with respect to packaging and encapsulation?

When passing data around a program, a common pitfall in development is relying on user input for a majority of function and method arguments. Classes solve this issue by encapsulating data relevant to the class, so methods need only to rely on absolutely necessary user input to modify or otherwise work with data. Encapsulating said data and only allowing users to modify it to your discretion improves not only the security of a program, but the usability (i.e., the program keeps track of important data and states rather than the user having to do it themselves).

6. How challenging was assignment 4 for you?

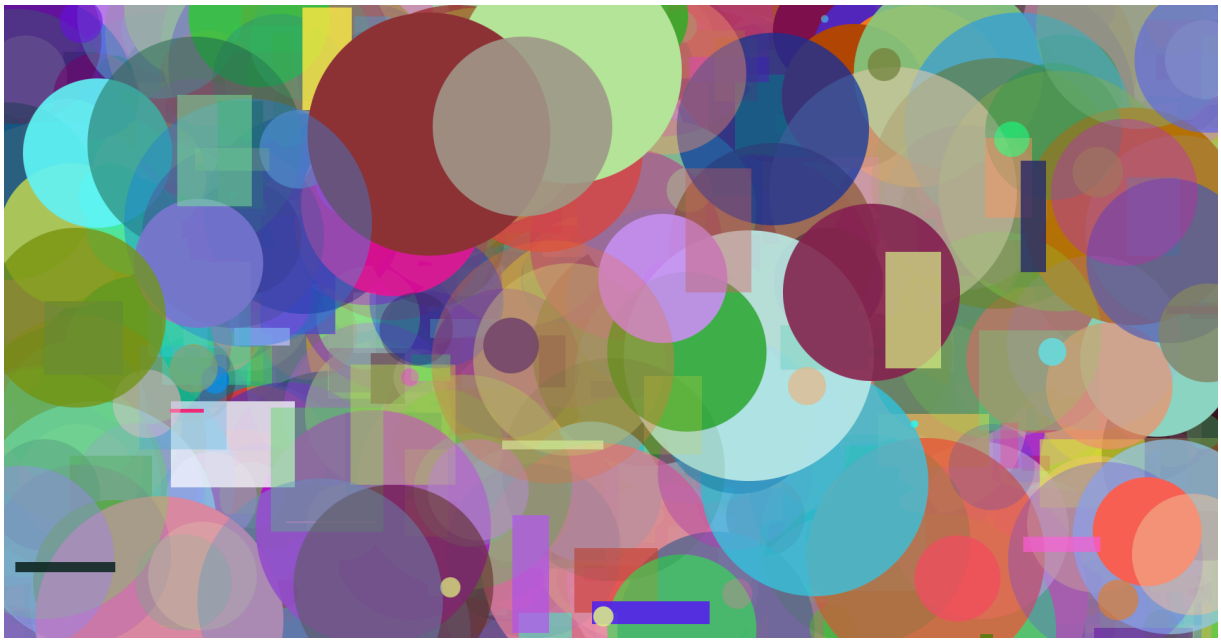
Admittedly, assignment 4 was slow to start. Upon beginning the assignment and reading the criteria I was initmidated due to how little explicit specification we were given. However, I eventually came to realize that the assignment wasn't just an exercise of our technical knowledge regarding HTML/SVG and Python, but rather our ability to solve complex problems on a medium scale, independently.

As soon as I came to this realization, I was able to breeze through the rest of the assignment, worrying less about exact criteria and more about what *I* think a quality solution should look like.

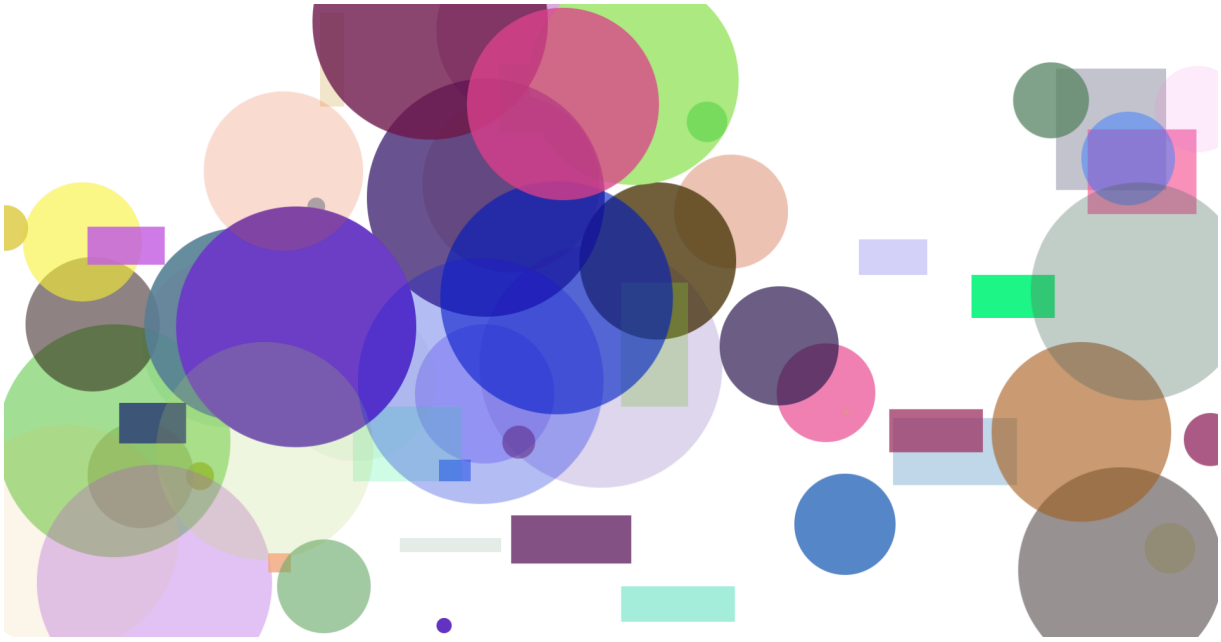
The format of this assignment is, relative to the other three, most realistic and simmilar to a task I might come across in the field. Due to this, I believe assignment 4 is the most valuable representation of our understanding of the course material, and our ability to problem solve in a work environment.

7. Did you send your artwork to your family?

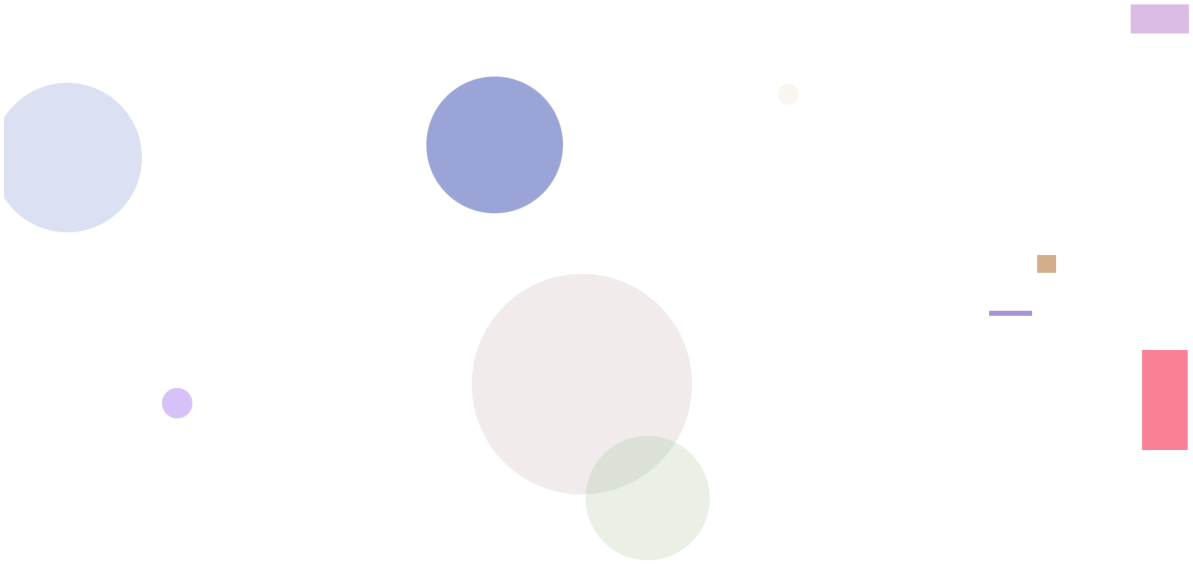
> I did! Although, I'm the first in my family to seek higher education in a tech field so I can't say they really knew much about what I was talking about. They liked the colours though! Attached, see a few examples of what the final prduct looks like :]



./a43.py 20 output.html 5000 960x500



./a43.py output.html 64 960x500



./a43.py output.html 10 960x500

8. Describe your continued learning experience in SENG 265.

SENG 265 is a complex mosaic of hugely important software development concepts and techniques - many of which are imperative in the vast majority of relevant career paths. SENG 265 was a valuable opportunity to dive into these topics in-depth and further develop my understanding of useful tools and practices, if not introducing entirely new concepts for the first time.

Even though I was previously familiar with a number of the topics covered in class, at no point in the course did I feel as though there was nothing more to learn. I live for developing my understanding of that which I'm passionate about, and this course was an extremely rich and exciting way to indulge in that passion in a relatively low-stakes environment, where I was free to take risks, be creative, and make mistakes without inhibiting the work of others.

I look forward to future software development classes and I truly hope similar classes allow for similar opportunities for exploration and experiential learning. Once again, my favourite activity in life is learning new things; I am forever grateful for the life I was born into, and the opportunities I was presented with both academically and socially (I know not everyone is as fortunate as we all are at UVIC!). I'm going to strive to do everything I can in the present and future to continue learning - that is, not only about tech, but about culture, accessibility, and all else that I might be unfamiliar with. Hopefully one of these days I'll be able to make significant change in the world and help those who need it most. At the moment, I'll learn all I can!

(my-label)=

Citations:

- [1] tldp (https://tldp.org/HOWTO/Secure-Programs-HOWTO/history.html)
- [2] opensource.com (https://opensource.com/19/9/command-line-heroes-bash)
- [3] initalcommit.com (https://initialcommit.com/blog/Technical-Guide-VCS-Internals)
- [4] wikipedia (https://en.wikipedia.org/wiki/Comparison_of_programming_languages)
- [5] lecture 12 slides (https://bright.uvic.ca/d2l/le/content/194492/viewContent/1619956/View)