Software Development Methods and Tools: A Learning Journey

SENG 265 Term Portfolio Project

Student Name: Tanuj Dargan
Student ID: V01040822
Course: SENG 265 - Software Development Methods
University: University of Victoria
Semester: Spring 2025
Date: March 23, 2025

Part 1

Purpose of This Portfolio

This portfolio documents my learning journey through SENG 265: Software Development Methods and Tools. It serves as a reflection of my experiences, challenges, and growth throughout the course.

Initial Expectations and Goals

When enrolling in SENG 265, my primary goals were to:

- Develop proficiency in command-line tools and environments
- Gain practical experience with version control systems
- Enhance my programming skills in C and Python
- Learn industry-standard software development methodologies

My Background in Software Development

Prior to this course, my experience with software development included:

- Languages: Python, Java, C++, JavaScript, TypeScript, R, Bash/Linux, SQL, LATEX
- Projects: Ray-Traced Environment in C++ using Hazel Game Engine | Plantbay using Python, Django | UVIC CourseMap and WebScraper using Next.js, Tailwind CSS, MongoDB, Python
- Frameworks/Tools: Familiar with Node.js, Next.js, React, Three.js, Vite, Django, Tailwind CSS, PyTorch, Tensorflow, OpenCV Pandas, Flask, Streamlit, GitHub, Git, Docker, AWS Suite

---

1. Weekly Learning Journal

Week 1: Jan 13–Jan 19: Learned basic Unix commands for navigating directories, managing files, and remote login to UGLS. Initially struggled with adapting to the command line and remembering commands but successfully connected to UGLS and completed file operations like copying, moving, and deleting.

Week 2: Jan 20–Jan 26: Practiced Vim for file editing and set up Git on Unix/Windows. Faced difficulties with Vim keybindings and Git push errors when switching OS, but eventually became comfortable editing files and managing repositories with Git commands like clone, add, commit, and push.

Week 3: Jan 27–Feb 2: Compiled C programs using gcc and structured code for Assignment 1 while introducing makefiles for build automation. Debugging compiler errors and organizing multiple source files were challenging at first, but troubleshooting skills improved, leading to better code management and execution.

Week 4: Feb 3–Feb 9: Worked with multi-file C projects, pointers, and wrote a factorial program. Faced segmentation faults when dealing with pointer arithmetic but gradually improved debugging skills. Successfully linked multiple files and ensured proper function calls using pointers, resulting in a well-structured program.

Week 5: Feb 10–Feb 16: Started with Python, learning basic syntax and using pandas for large dataset analysis. Encountered issues with Python's dynamic typing and CSV performance but created a script to filter and analyze F1 race-winner data efficiently. Appreciated Python's ease of use compared to lower-level languages.

Week 6: Feb 24–Mar 2: Explored advanced Python concepts, including tuples, dictionaries, and the datetime module for date calculations. Managed complex nested data structures and worked through challenges with time conversions. Successfully implemented a feature to track upcoming lab dates, improving both efficiency and readability in code.

Week 7: Mar 3–Mar 9: Implemented linked lists in C, handling insertion, traversal, and deletion while ensuring dynamic memory allocation with malloc. Faced challenges with memory leaks and segmentation faults but refined debugging techniques. Also improved makefile usage for multi-file compilation, leading to more organized and maintainable code.

Week 8: Mar 10–Mar 16: Focused on Python test automation by writing unit tests and using assertions to verify function correctness. Faced challenges in structuring tests for multiple cases and handling edge scenarios but successfully created a robust testing suite. Implementing automated testing streamlined debugging and helped prevent regressions.

---

2. What is the core functionality of Jupyter Notebooks?

Core Functionalities:

- **Notebook Document:**
  Jupyter Notebook documents combine executable code (e.g., Python) with rich text elements like paragraphs, equations, figures, and links. They are both human-readable and executable, enabling seamless coding, analysis, and documentation in one place.

- **Language Support:**
  Jupyter supports over 40 programming languages, including Python, R, Julia, and Scala, making it a versatile tool for data science, machine learning, and computational research.

- **Sharing Capabilities:**
  Notebooks can be shared via email, Dropbox, GitHub, and Jupyter Notebook Viewer, allowing for easy collaboration and knowledge exchange.

- **Interactive Output:**
  Jupyter Notebooks generate rich, interactive outputs, including HTML, images, videos, LaTeX equations, and custom MIME types, enhancing data presentation and visualization.

- **Extensions:**
  Jupyter supports extensions that enhance functionality, such as "Table of Contents" for navigation and "Variable Inspector" for viewing active variables, improving workflow efficiency.

- **Data Analysis and Visualization:**
  Jupyter is widely used for data cleaning, transformation, numerical simulation, visualization, machine learning, and deep learning, making it a comprehensive tool for data science.

- **Open-Source and Free:**
  As an open-source web application, Jupyter is free to use, modify, and distribute, with strong community support for continuous improvements and accessibility.

---

3. Simple Jupyter Notebook Markdown (Including Links and Images)

Markdown is used in Jupyter Notebooks to format text cells. Below are the essentials:

1. **Headings**

   # Heading 1

   ## Heading 2

   ### Heading 3

   More hash marks indicate deeper levels of headings.

2. **Lists**

   - Bulleted lists use asterisks or dashes:
     - Item 1
     - Item 2
   - Numbered lists use numbers:

     A. First
     B. Second

3. **Links**
   Use the bracket and parenthesis syntax:

   Link Text

4. **Images**
   The syntax matches links but with an exclamation mark in front:

   ![image description] (https://www.example.com/image.jpg) 

4. How can you typeset mathematical formulas including Greek letters using LaTeX Markdown in Jupyter Notebooks? LaTeX is used extensively used for documents with mathematical formulas.

Jupyter Notebooks support LaTeX for mathematical formulas:

- Inline equations use single dollar signs: $E = mc^2$ The input is `$E = mc^2$`
- Display equations use double dollar signs:

$$\frac{d}{dx}(x^n) = nx^{n-1}$$

Input LaTeX: `$$\frac{d}{dx}(x^n) = nx^{n-1}$$`

$$\int_a^b f(x)dx = F(b) - F(a)$$

Input LaTeX: `$$\int_a^b f(x) dx = F(b) - F(a)$$`

Greek letters are commonly used in formulas:

- Alpha: α
- Beta: β
- Gamma: γ
- Delta: Δ
- Pi: π Example Input for greek letters: `$\alpha$`

Matrix example:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

---

5. What is the history of Unix and Linux? How prevalent is Unix or Linux in software development today?

History of Unix and Linux

Unix, initially developed by Ken Thompson, Dennis Ritchie, and others at Bell Labs in the late 1960s and early 1970s, evolved through collaborations with Unix System Laboratories and Sun Microsystems. The release of BSD Unix in 1992 led to derivatives like NetBSD and FreeBSD. Linux, introduced in 1991 by Linus Torvalds, is an open-source Unix-like OS built around the Linux kernel. Inspired by MINIX, Linux is distributed with system software and libraries, often from the GNU Project, forming various Linux distributions.

Prevalence in Software Development Today

Unix and Linux dominate software development, replacing many proprietary Unix systems. Their presence extends beyond servers into consumer desktops, mobile devices, and embedded systems. Notable Linux distributions include Red Hat, Fedora, SUSE, Debian, and Ubuntu. Linux is particularly significant in server environments, often running without a graphical interface or using stacks like LAMP. Mac OS X, a Unix-based system, integrates BSD and the Mach kernel, further cementing Unix's influence in the consumer market.

Linux and Unix-based systems are essential in:

- Web servers and cloud infrastructure
- Mobile OS platforms like Android
- DevOps and system administration
- Scientific computing and data science

---

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

What is Bash? 

Bash (Bourne Again Shell) is the default command-line interpreter for most Unix and Linux distributions. It is a command processor that operates in a text window, allowing users to input commands to trigger actions. Bash serves as a command language interpreter and is the default on many GNU/Linux systems. The name is a play on words, referencing the Bourne shell it replaces and the concept of being "born again."

Core Functionality of Bash

Bash executes commands interactively or from a file, known as a shell script. A script comprises standard commands and advanced scripting constructs. Any command executable in a script can also be run directly in the terminal. Bash supports features such as filename globbing, piping, here documents, command substitution, variables, and control structures. It inherits elements from sh while incorporating enhancements from csh and ksh, making it a POSIX-compliant shell with extensions.

Bash Scripting

Bash scripting automates tasks, eliminating the need for manual execution. It is essential for:

- Automation and scripting
- System administration
- Software development and deployment

Bash scripts support variables, conditional statements, and loops, similar to programming languages. They are widely used for file manipulation, routine tasks (e.g., backups), and general automation, enabling the execution of multiple commands as a single command.

---

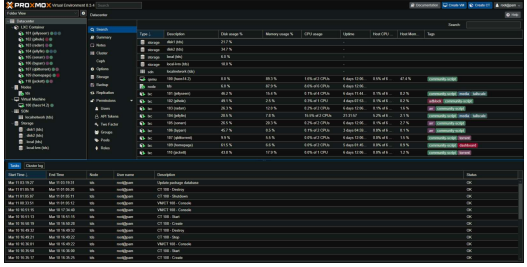
7. What is your background in the scripting language Bash? How prevalent is Bash in software development today?

My background with Bash

My background with Bash is extensive and rooted in practical, hands-on experience. I manage a homelab where I host numerous services using Proxmox, and Bash plays a critical role in automating and maintaining this environment. I've written scripts to automate service installations, container creation, and other repetitive tasks, which has significantly streamlined my workflow.

Beyond my personal projects, I actively contribute to the community by sharing these scripts, helping others set up and manage their own services efficiently. This has deepened my understanding of Bash, as I often troubleshoot and optimize scripts to ensure they work across different environments. My experience has taught me how powerful Bash can be for system administration, automation, and orchestration tasks.

Take a look at my Proxmox Homelab:



View My Github

How prevalent is Bash in software development today?

Bash remains a cornerstone of software development, particularly in environments that rely on Unix/Linux systems. It is widely used for automating tasks, managing system configurations, and orchestrating workflows. System administrators and DevOps engineers rely on Bash for tasks like server provisioning, container orchestration (e.g., Docker), and CI/CD pipelines. Developers also use Bash to automate repetitive tasks, such as compiling code, running tests, and deploying applications.

Bash is especially prevalent in homelabs, like mine, where it simplifies the management of services and infrastructure. It's also a beginner-friendly scripting language, making it an excellent starting point for those new to programming or system administration. While more versatile languages like Python are often preferred for complex tasks, Bash excels in scenarios where quick, lightweight automation is needed.

In summary, Bash continues to be a vital tool in software development and system administration, and my experience with it has been both extensive and impactful in my personal and community-driven projects.

---

8. How can Bash be used for automating tasks? Frequent midterm question.

Bash Scripting

Bash scripting automates tasks, eliminating the need for manual execution. It is essential for:

- Automation and scripting
- System administration
- Software development and deployment

Bash scripts support variables, conditional statements, and loops, similar to programming languages. They are widely used for file manipulation, routine tasks (e.g., backups), and general automation, enabling the execution of multiple commands as a single command.

How Can Bash Be Used for Automating Tasks?

Bash is widely used for task automation, reducing manual work and improving efficiency. Some common applications include:

- **Automating System Administration**: Automating software installations, system updates, log file analysis, and scheduled backups using cron jobs.
- **File and Directory Management**: Automating file organization, renaming, and cleaning up directories.

- **Data Processing**: Extracting, filtering, and transforming data using commands like `awk`, `sed`, and `grep`.
- **Deployment Automation**: Automating code deployment, build processes, and CI/CD pipelines.
- **Monitoring and Logging**: Automating system health checks and alerting administrators of critical events.

---

9. Why are version control and configuration management critical in software development projects? Frequent midterm question.

Version control and configuration management are essential for maintaining code integrity, collaboration, and consistency in software development projects.

Version Control

- Tracks changes to code, enabling rollback to previous versions if needed.
- Facilitates collaboration among multiple developers, preventing conflicts.
- Provides a history of modifications for debugging and auditing.
- Supports branching and merging, allowing parallel development efforts.
- Examples: Git, Subversion (SVN), Mercurial.

Configuration Management

- Ensures consistency across different environments (development, testing, production).
- Automates software setup and dependency management.
- Reduces errors caused by manual configuration changes.
- Enhances scalability by maintaining infrastructure as code (IaC).
- Examples: Ansible, Puppet, Chef, SaltStack.

Both version control and configuration management contribute to efficient development workflows, reducing risks and increasing software reliability.

---

10. What are the core functionalities of Git, Github, and Gitlab? Frequent midterm question.

Git

Git is a distributed version control system designed to track changes in files and coordinate work among multiple developers. It allows users to manage file revisions, branch and merge efficiently, and work offline. Git is open-source, enabling users to modify its source code, and is widely used for collaborative software development due to its speed, reliability, and support for non-linear workflows.

GitHub

GitHub is a cloud-based hosting service for Git repositories. It extends Git's functionality by providing tools for collaboration, project management, and deployment. GitHub allows developers to push and pull changes, manage repositories, and collaborate globally. It also includes features like pull requests, issue tracking, and actions for CI/CD workflows, though its security features are considered less robust compared to GitLab.

GitLab

GitLab is a comprehensive DevOps platform that integrates Git repository hosting with tools for project planning, monitoring, and security. It supports both open and private repositories, issue tracking, wikis, and CI/CD pipelines. GitLab is known for its enhanced security features, support for various attachments, and organization-wide permissions. It streamlines development workflows and reduces product lifecycles, making it a powerful tool for teams.

---

11. How is Git used as a version control system?

Git is used as a version control system to track changes in files and manage collaborative development. It allows developers to create repositories where they can store and organize code. Key functionalities include:

- **Version Tracking**: Git records every change made to files, enabling developers to view the history of modifications and revert to previous versions if needed.
- **Branching and Merging**: Developers can create branches to work on features or fixes independently, then merge them back into the main codebase without disrupting others' work.
- **Collaboration**: Git enables multiple developers to work on the same project simultaneously by managing conflicts and integrating changes efficiently.
- **Distributed Workflow**: Each developer has a complete copy of the repository, allowing them to work offline and sync changes later.
- **Commit History**: Git maintains a detailed log of changes, including who made them and why, providing transparency and accountability.

Git's flexibility and efficiency make it an essential tool for managing codebases in software development.

---

12. How can Git be used effectively in a highly distributed software development environment involving multiple countries?

Git is highly effective in distributed environments due to its decentralized nature, allowing developers across different countries to collaborate seamlessly. Each developer has a complete copy of the repository, enabling offline work and asynchronous contributions, which is ideal for teams in different time zones.

Key features like branching and merging allow developers to work on isolated features or fixes without disrupting the main codebase. Remote repositories (e.g., GitHub or GitLab) act as central hubs for syncing changes, while pull requests and code reviews ensure high-quality contributions. Git’s conflict resolution tools help integrate changes smoothly, and its detailed commit history provides transparency and accountability.

Additionally, Git integrates with CI/CD pipelines to automate testing and deployment, ensuring a stable codebase. These features make Git an essential tool for managing code in globally distributed teams.

---

13. What is your background in the programming language C? How prevalent is C in software development today?

My background with C

My experience with C comes from working on embedded systems and low-level programming projects. As part of the UVic Formula Hybrid Team, I developed sensor systems for a hybrid car using the CAN communication protocol in C, ensuring efficient data integration and control. This hands-on experience has given me a strong understanding of C's role in hardware-level programming and real-time systems. Additionally, my academic background in computer science has reinforced my knowledge of C's syntax, memory management, and performance optimization.

How prevalent is C in software development today?

C remains highly relevant in software development, especially for systems that require high performance and direct hardware interaction. It is widely used in operating systems, embedded systems, game engines, and other performance-critical applications. Despite the rise of higher-level languages, C's efficiency, portability, and control over system resources make it indispensable for foundational software. Many modern languages, like C++ and Python, are influenced by C, further cementing its importance in the software development ecosystem.

---

14. What is your background in the programming language Python 3? How prevalent is Python 3 in software development today?

My background with Python 3

I have extensive experience with Python 3, using it across various projects and roles. At Nanodock, I developed workflows using Python, Docker, and NixOS to generate developer environments. During my research internship at Makers Asylum, I analyzed water contamination data with Python and Excel. Additionally, I've built Python-based tools like a web scraper for UVic CourseMap, optimized with BeautifulSoup, and a Django app for analyzing plant health. My work also includes developing a quantitative finance library and participating in hackathons, where I used Python for machine learning and reinforcement learning models.

How prevalent is Python 3 in software development today?

Python 3 is one of the most widely used programming languages today due to its simplicity, versatility, and extensive libraries. It is prevalent in fields like web development, data science, machine learning, automation, and scientific computing. Python's readability and ease of use make it a popular choice for beginners, while its powerful frameworks and tools make it indispensable for professionals. Its adoption in academia, startups, and large enterprises ensures its continued relevance in modern software development.

---

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

The most popular programming languages today as per the 2024 Stackoverflow Developer Survey include Python, JavaScript, Typescript, Java, C, and C++. Python is favored for its simplicity, versatility, and extensive libraries, making it ideal for data science, machine learning, and web development. JavaScript dominates web development due to its ability to create interactive frontends and backends.

Java is widely used for enterprise applications and Android development because of its stability and scalability. C and C++ are essential for systems programming, game development, and performance-critical applications due to their speed and low-level control. These languages remain popular because they address diverse needs across industries and have strong community support.

---

16. C & Python are among the most popular programming languages. You are learning them in SENG 265. How cool is that for your software development career?

Learning C and Python in SENG 265 is incredibly beneficial for your software development career. C provides a strong foundation in low-level programming, memory management, and performance optimization, which are critical for systems programming and embedded systems. Python, on the other hand, is versatile and widely used in fields like data science, machine learning, and web development. Mastering both languages equips you with a diverse skill set, enabling you to tackle a wide range of projects and making you highly competitive in the software development industry.

---

17. Describe the fundamental differences between C and Python.

Feature	C	Python
Type System	Static typing	Dynamic typing
Memory Management	Manual memory allocation/deallocation	Automatic garbage collection
Syntax	Verbose, requires semicolons	Concise, uses indentation for blocks
Compilation	Compiled to machine code	Interpreted (bytecode)
Performance	Generally faster execution	Generally slower execution
Development Speed	Slower development cycle	Rapid development
Use Cases	Systems programming, embedded systems	Web development, data science, automation
Libraries	Standard library is minimal	Extensive standard library and third-party packages

C and Python differ fundamentally in their design and use cases. C is a low-level, statically typed, compiled language that provides fine-grained control over memory and system resources, making it ideal for systems programming and performance-critical applications. Python, on the other hand, is a high-level, dynamically typed, interpreted language known for its simplicity and extensive libraries, making it perfect for rapid development, data science, and automation. While C prioritizes performance and efficiency, Python emphasizes ease of use and developer productivity. Both are powerful tools, excelling in different domains of software development.

---

18. How challenging was learning C for you?

Learning C was slightly difficult for me initially because its syntax and manual memory management are quite different from higher-level languages like Python. However, I found it rewarding as it gave me a

deeper understanding of how computers work at a low level. Working on projects like developing sensor systems for the UVic Formula Hybrid car using C helped me appreciate its power and efficiency, making the learning process worthwhile.

---

19. How challenging was completing Assignment 1 for you?

Completing Assignment 1 was a bit difficult as I frequently confused C's syntax with other languages I know, like Python. Debugging and understanding manual memory management also took some time. However, AI tools like Claude were incredibly helpful in clarifying concepts and troubleshooting errors. Despite the challenges, the assignment was a great learning experience and helped me strengthen my understanding of C. GPT-4o AI agent thumbnail

---

20. Required: What are your personal insights, aha moments, and epiphanies you experienced in the first part of the SENG 265 course?

In the first part of SENG 265, I had several insights and "aha" moments that deepened my understanding of programming and problem-solving. One key realization was how foundational concepts like memory management and pointers in C provide a deeper understanding of how computers operate at a low level. This was an eye-opener, as I had previously relied on higher-level languages like Python, which abstract these details. Another epiphany came when I successfully debugged a segmentation fault for the first time—it taught me the importance of careful memory handling and how small mistakes can have significant consequences. Additionally, learning to write efficient, concise code in C helped me appreciate the balance between performance and readability. These moments not only enhanced my technical skills but also shifted my perspective on programming, making me more detail-oriented and methodical in my approach.

---

21. Required: How did you experience GenAI tools (e.g., OpenAI chatGPT, Microsoft CoPilot, Google Gemini, Perplexity) as learning tools?

I found GenAI tools, particularly Claude, to be incredibly helpful as learning aids during the course. Claude provided clear explanations of complex concepts, helped debug code, and offered alternative approaches to solving problems. It was especially useful when I struggled with C syntax or debugging errors, as it provided step-by-step guidance that enhanced my understanding. Unlike static resources, Claude's interactive nature allowed me to ask follow-up questions and refine my learning process. It acted as a supportive tutor, helping me bridge gaps in my knowledge and boosting my confidence in tackling assignments. Overall, using Claude made learning more efficient and engaging.

---

22. Record your Generative AI prompts for Bash, C, and Python inquiries for easy recall.

Bash:

- "How to use scp and basic UNIX commands to interact with ssh external servers?"
- "Key `vim` commands and Git setup steps on UNIX/Windows?"

C:

- "How to compile and link C programs using `gcc` and makefiles?"
- "Explain dynamic memory allocation and linked lists in C."
- "Example of using `qsort` in C to sort random values."
- "Steps to process CSV data in C and generate output files."

Python:

- "Basics of Python: pandas, loops, lists, and dictionaries?"
- "How to analyze CSV data with pandas (e.g., F1 race winners)?"
- "Write Python code for unit testing and assertions."
- "Use `datetime` to calculate remaining lab days."

Part 2

---

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

I find SENG 265 extremely valuable for my resume because it proves I can handle both systems-level and high-level programming. Knowing C and Python together makes me more versatile, and it shows potential employers that I have a solid understanding of memory management and lower-level concepts, while also being able to write high-level scripts for automation or data handling. I feel confident listing SENG 265 on my resume and highlighting projects from the class to demonstrate my practical skills. I also believe that Assignments 1 to 3 were highly relevant to real-world job tasks, as they involved analyzing large datasets, which is a common aspect of many professional co-op roles.

---

2. In SENG 265 you acquired significant systems programming knowledge. You learned many different skills (e.g., C, Python, Unix, Bash, make, pipes & filters, SVG, HTML, static and dynamic data structures, testing and debugging, logging, text processing,decorators, GenAI tools and many more). Which skills are most valuable for you? Which skills are you going to add to your resume?

For me, the most valuable skills I gained are definitely C programming and Python. They show that I can code close to the hardware (with C) and also move quickly in higher-level scripting tasks (with Python). I also really value my newfound comfort with Unix and Bash, because command-line proficiency is often essential for DevOps tasks and software testing pipelines. I plan to feature these skills on my resume, along with testing and debugging experience, because they showcase my ability to write reliable code. I'll also mention pipes and filters and make, since they illustrate that I can handle build processes and the

Unix toolbox effectively. I would also mention my expertise gained with Git as its a highly important and useful version control and collaboration tool used by most organisations.

---

3. Describe the notion of and motivation for typing and typing hints in Python. The strongly typed programming language Rust is rapidly being adopted by companies and is used to improve the performance of Python. Why is Rust getting so popular? Does typing play a role?

In my view, Python typing hints give me a clearer way to show what type of data I expect in a function or a variable. Even though Python doesn't strictly enforce these types at runtime, using hints makes my code more understandable to both coworkers and automated tools like mypy or IDEs. It helps catch errors early, improving the reliability of my projects. As for Rust, I believe companies are excited about it because it offers memory safety without the overhead of a garbage collector. It can potentially replace performance-critical parts of Python applications, which is a big deal when scaling. Rust has a strong type system, and while typing is part of its appeal, I think the real magic is in how Rust manages memory safely through ownership and borrowing, which appeals to organizations that need both speed and safety. The safe memory mangement also makes it a noteable replacement for C/C++ given its performance compared to python.

---

4. Describe the Python concepts of slicing and comprehension. Can all loops be implemented using slicing and comprehensions?

Slicing is how you would extract segments of lists, strings, or tuples using a [start:end:step] notation. It's concise and super useful for quick data manipulation. Comprehensions offer a short way to build or transform lists, dictionaries, or sets—like [x*x for x in range(10)]. Although comprehensions and slicing can replace a lot of simple loops, I can't do absolutely everything with them (for instance, loops that involve multiple complex steps, external function calls, or intricate branching logic might be more readable in an explicit loop). But for data transformation, slicing plus comprehensions often lets me write succinct, Pythonic code.

---

5. What is the difference between pytonic and non-pytonic code?

When I write Pythonic code, I use idiomatic Python features like list comprehensions, context managers, meaningful variable names, and I follow PEP 8 style guidelines. It makes my code more readable and elegant. Non-Pythonic code might look more like a direct translation from a lower-level language into Python, or it might ignore Python's conventions and best practices. It often appears verbose or clunky, missing the simplicity and clarity Python encourages.

---

6. Describe the concepts and applications of instance & class variables and instance & class methods.

- **Instance variables** belong to individual objects, and each object can hold different values for these variables.
- **Class variables** are shared across all instances of the class.
- **Instance methods** require a self parameter, letting me operate on an individual object's data.
- **Class methods** take a cls parameter instead, and usually let me perform actions or create alternative constructors at the class level.

I love how this design pattern helps me structure data in a way that's intuitive and easy to maintain.

---

7. Describe the concepts of the Python logging module.

The logging module lets me log messages in a structured way, with log levels like DEBUG, INFO, WARNING, ERROR, and CRITICAL. By using logs instead of print statements, I can easily toggle the verbosity of my program or redirect logs to a file or external service. This module keeps my code cleaner and helps me debug or monitor my application in production environments.

---

8. Describe the importance of implicit or tacit knowledge in requirements engineering and job interview dialogues (i.e., interviewers want to know whether you are able to ask question) as discussed in class.

I've noticed that not every requirement is spelled out. Sometimes, there's unspoken "domain" knowledge that I have to uncover by asking questions or observing how people use the system. In interviews, hiring managers often want to see if I can spot incomplete specs or ambiguous issues. By asking clarifying questions, I show that I understand real-world complexities, where not everything is neatly documented. This is especially important in requirements engineering, where missing or overlooked details can derail entire projects. A simple example of this would be when an interviewer uses an Acronym that I may not be familiar with or may know but may have a different idea in mind. By asking to clarify such terms I would be able to express how I can get around real world complexities of an engineering environment.

---

9. Describe the Python concepts of deep copy and shallow copy of objects as well as the purpose of the methods **copy**() and **deepcopy**() as outlined in the Python copy module.

- A **shallow copy** creates a new object but still references nested objects from the original. So if I change a nested object in the original, the copy is affected too.
- A **deep copy** recursively copies all objects, so each nested structure is cloned and changes don't propagate between copies.
- The methods **copy**() and **deepcopy**() let me customize how my objects should be copied, which is handy if I have special reference handling or performance considerations.

---

10. Python classes package and encapsulate data and functions that operate on those data. What are the big

advantages of classes with respect to packaging and encapsulation? What are the big advantages of object-oriented design?

Classes offer me clear packaging of data (attributes) and behaviors (methods), which helps keep my code organized. Encapsulation means I can change the internal implementation without breaking the external interface. This leads to better maintainability. With object-oriented design, I can reuse code using inheritance, or create flexible designs that rely on polymorphism. It encourages modular thinking, so each class has a well-defined role—this ultimately speeds up development and reduces the likelihood of spaghetti code in larger projects.

---

11. How challenging was Assignment 2 for you?

Assignment 2 significantly easier for me as I had worked with the panads library prior to this course. However, the most helpful factor was the fact that python introduced a lot of easy to use and developer friendly syntax alongside concepts. Making loop and function syntax helped make life easier by making the coding part easier. I realized how Assignment 1 set a strong base for what was to come in Assignment 2 and future assignments as well. I think having an understanding of how to perform the operations in C made it easy to adapt to Python for this assignment.

---

12. How challenging was Assignment 3 for you?

Assignment 3 was a bit of a steeper learning curve because I was still getting comfortable with C syntax and memory management. Debugging was tricky, but once I got the hang of it, I realized how valuable the process was for building my confidence in systems programming. The switch from Python to C was significant but it helped keep my knowledge of C intact while giving me experience in Python.

---

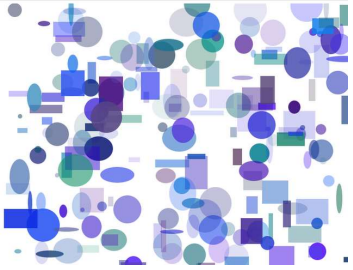
13. How challenging was Assignment 4 for you?

Assingment 4 was a step away from the likes of Assignments 1,2 and 3. Given the different requirements and rather more open expectations in terms of the output for each part of the assignment, I found it a bit hard to know whether I was on track. However, after I finished part 1, I was able to grasp a good idea of what the overall goal of the assignment was and what it was trying to teach me. As soon as that understanding set in, I was more confident and felt like part 2 and 3 were easier to do even if they were relatively more technically challenging.

---

14. Did you produce some impressive SENG 265 Art with Assignment 4?

Generating custom art required me to connect everything from text processing to geometry. In the end, I was proud of the artwork I created; it felt like a great fusion of code and creativity. I enjoyed the step away from output logs and csv files to something more visually appealing.



---

15. Did you show off your artwork generated with your Assignment 4 solution to your friends and family?

Yes, I did! It was fun explaining how I used programming to produce visual art. They seemed extremely impressed with the connection to maths, geometry and art that the assignment built.

---

16. [Required] Summarize PEP 8 and be prepared to answer questions about PEP 8 in the final exam.

PEP 8 is Python's style guide that emphasizes readability and consistency. It addresses indentation (usually four spaces), recommends keeping line lengths to a reasonable limit, and covers naming conventions (like snake_case for variables and methods, CamelCase for class names). It also sets standards for whitespace around operators, grouping imports, and adding clear code comments. By following PEP 8, I keep my code looking familiar to other Python developers and make it easier to maintain.

---

17. [Required] Describe at least five of your favourite libraries in the Python ecosystem -- in addition to the libraries you described for TPP Part 1.

Five of My Favorite Python Libraries: PyTorch, OpenCV, Flask, Streamlit, and Beautiful Soup

1. PyTorch

Domain: Machine Learning and Deep Learning
PyTorch is one of the most popular deep learning frameworks, developed by Facebook's AI Research lab. It provides a flexible and dynamic computational graph, making it easier to build and train neural networks. PyTorch is widely used for research and production in machine learning and AI. I enjoy using PyTorch because it feels intuitive for creating and training neural networks for deep learning. It uses a dynamic computational graph, so I can experiment and debug my models as I go without a lot of boilerplate code. PyTorch is backed by a huge community, and I find plenty of tutorials and ready-made components to jumpstart my machine learning projects. It's also great for research-level projects where I need maximum flexibility and frequent changes to model architectures.

- **Key Features:**
  - Dynamic computation graphs allow for flexibility during model development.
  - Built-in support for GPU acceleration, enabling faster training of models.
  - Extensive libraries for computer vision (TorchVision), natural language processing (TorchText), and more.

2. OpenCV

Domain: Computer Vision
OpenCV (Open Source Computer Vision Library) is a powerful library for computer vision and image processing tasks. It supports a wide range of operations, from basic image manipulation to advanced computer vision algorithms like object detection and facial recognition. OpenCV is my go-to for computer vision tasks. I like it because it has a wealth of pre-built tools for image manipulation, object detection, and more advanced features like facial recognition. Whenever I need to process large batches of images, create visual effects, or prototype something like a gesture-control system, OpenCV's performance and breadth of features come in handy. It saves me from having to reinvent the wheel for most classic computer vision algorithms.

- **Key Features:**
  - Tools for image and video processing, including filtering, transformations, and feature detection.
  - Pre-trained models for tasks like face detection, object tracking, and optical flow.
  - Cross-platform support and integration with other libraries like NumPy and PyTorch.

3. Flask

Domain: Web Development
Flask is a lightweight and minimalist web framework for Python. It is designed to be simple and flexible, allowing developers to build web applications quickly without unnecessary overhead. Flask is often used for small to medium-sized projects or as a backend for APIs. For lightweight web applications and APIs, Flask is fantastic. It's a micro-framework, so I can keep my application structure simple—perfect for smaller projects or quick prototypes. At the same time, Flask is so flexible that I can scale up and add extensions for databases, authentication, or session management if I need them. I love how little overhead it imposes, letting me focus on core functionality first.

- **Key Features:**
  - Minimalistic and modular design, giving developers full control over the application structure.
  - Built-in development server and debugger for rapid prototyping.
  - Extensive ecosystem of extensions for adding features like authentication, database integration, and more.

4. Streamlit

Domain: Data Science and Interactive Applications
Streamlit is a modern library for creating interactive web applications for data science and machine learning projects. It allows developers to turn Python scripts into shareable web apps with minimal effort. When I want to turn a data project or machine learning prototype into a shareable web app fast, Streamlit is my best friend. I can write a few lines of Python, use Streamlit's built-in components for sliders or text fields, and have an interactive dashboard running almost instantly. It's especially useful if I need to show stakeholders a visual demo of my model predictions or data transformations—no need to write front-end code or learn a more complex web framework.

- **Key Features:**
  - Simple syntax for creating interactive widgets like sliders, dropdowns, and charts.
  - Real-time updates to visualizations and data as users interact with the app.
  - Seamless integration with popular data science libraries like Pandas, Matplotlib, and Plotly.

5. Beautiful Soup

Domain: Web Scraping
Beautiful Soup is a library for parsing HTML and XML documents, making it easy to extract data from web pages. It is widely used for web scraping projects and works well with other libraries like Requests for fetching web content. For web scraping, I gravitate toward Beautiful Soup. It's straightforward and excels at parsing HTML, letting me find and extract the data I need without fuss. If I combine it with libraries like requests, I can quickly gather information from websites for personal projects, data analysis, or even building small-scale crawlers. Its simple search methods (like find and find_all) are easy to remember and cover most use cases I run into.

- **Key Features:**
  - Intuitive methods for navigating, searching, and modifying the parse tree.
  - Support for different parsers, including Python's built-in HTML parser and third-party parsers like lxml.
  - Handles poorly formatted HTML gracefully, making it robust for real-world web scraping.

---

18. [Required] Document your continued learning experience in SENG 265.

As I covered upto week 8 in Part 1:

Week 9: Focused on Regex in Python where we learned to read files and as an exercise completed extraction of data from logs such as date, package name and more. This week also comprised of Assignment 4 where I got to combine my artistic side with math and code to create cool artwork using geometry and random generation of numbers. Majority of the course's learning experience was covered in Part 1.

Week 10: This week covered midterm solutions, list comprehensions, string methods, and embedding Python in Bash. The focus then shifted back to C, exploring structs and dynamic memory allocation. The week concluded with an introduction to dynamic data structures, specifically singly and doubly linked lists (SLL, DLL).

Week 11: Building on dynamic data structures, this week introduced defensive programming, doubly linked lists, and tree structures. The discussion expanded to graphs, algorithms, and Python decorators, with practical implementations in C and Python. The week ended with software engineering principles, covering object-oriented concepts such as "IS-A" and "PART-OF" relationships, along with classes, instances, and constructors in Python.****

---

19. [Required] How did you experience Generative AI (e.g., chatGPT, Gemini, Perplexity, or Copilot) as a learning tool? This answer must be different from what you answered in TPP1 (i.e., the assumption is that you got to know genAI tools better in the first part of the course and you really used it in the second part of the course).

In the second half of the course, I began to see GenAI tools as active collaborators. Instead of just searching for quick answers, I used them to brainstorm approaches or debug tricky errors in smaller code snippets. I also learned to be more skeptical and verify the suggestions I got from AI. That shift from passive consumption to thoughtful collaboration was a big deal for me, and it made me feel more

in charge of how I use these tools. I began to use AI integrated IDE's like Cursor that offer a way more intuitive experience with LLM's to help me debug my code and manage proper documentation in case anything is misrepresented or doesnt match PEP8 and other standards.

---

20. [Required] What are your personal insights, aha moments, and epiphanies you experienced in the first part of the SENG 265 course?

One of my biggest insights was how quickly minor mistakes in C can lead to major debugging headaches. That taught me the value of writing robust tests and carefully thinking about memory usage. Another aha moment was realizing how much faster I could develop certain features in Python compared to a lower-level language. Finally, I had an epiphany about the value of clarity in communication—both in code (through logging, comments, or docstrings) and in asking questions whenever requirements were unclear.

References and Bibliography

Course Materials

- SENG265, Dr. Hausi A. Müller, University of Victoria, 2025 Spring
- Course slides and lecture notes from SENG265 Brightspace

Online Resources

- Python Software Foundation. (n.d.). *Python Documentation*. Retrieved from https://docs.python.org/3/
- Git Project. (n.d.). *Git Documentation*. Retrieved from https://git-scm.com/doc
- Jupyter Project. (n.d.). *Jupyter Notebook Documentation*. Retrieved from https://jupyter-notebook.readthedocs.io/en/stable/notebook.html
- Red Hat. (n.d.). *A Brief History of Unix and Linux*. Retrieved from https://www.redhat.com/sysadmin/unix-linux-history
- DigitalOcean. (n.d.). *A Brief History of Linux*. Retrieved from https://www.digitalocean.com/community/tutorials/brief-history-of-linux
- Devhints. (n.d.). *Bash Cheatsheet*. Retrieved from https://devhints.io/bash
- Tables Generator. (n.d.). *Markdown Tables Generator*. Retrieved from https://www.tablesgenerator.com/markdown_tables
- FrontPageLinux. (n.d.). *Guide Through the History of Unix/Linux*. Retrieved from https://frontpagelinux.com/articles/guide-through-history-of-unix-linux-everything-you-need-to-know/
- FreeCodeCamp. (n.d.). *Git and GitHub Crash Course for Beginners*. Retrieved from https://www.freecodecamp.org/news/git-and-github-for-beginners/
- Towards Data Science. (n.d.). *Mastering Jupyter Notebooks*. Retrieved from https://towardsdatascience.com/mastering-jupyter-notebooks-f6a1db30c43f
- Linux Handbook. (n.d.). *Essential Linux Commands*. Retrieved from https://linuxhandbook.com/linux-commands-cheat-sheet/
- Beautiful Soup Richardson, L. (2007). Beautiful Soup Documentation. Retrieved from https://www.crummy.com/software/BeautifulSoup/

- Streamlit Inc. (2019). Streamlit: The Fastest Way to Build and Share Data Apps. Retrieved from https://streamlit.io
- Ronacher, A. (2010). Flask: Web Development, One Drop at a Time. Retrieved from https://flask.palletsprojects.com
- Bradski, G. (2000). The OpenCV Library. Dr. Dobb's Journal of Software Tools. Retrieved from https://opencv.org
- Paszke, A., Gross, S., Massa, et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. Advances in Neural Information Processing Systems, 32. Retrieved from https://pytorch.org

GenAI-Assisted Content

- Claude 3.5 Sonnet, Claude 3.7 Extended, OpenAI o1, ChatGPT
- You.com AI Search
- Claude AI Platform (https://claude.ai)
- OpenAI ChatGPT (https://chat.openai.com/)