

Term Portfolio Project

Matthew Goosney

University of Victoria

SENG-265, Spring 2025

Table of Contents

1. Introduction to C: C Lectures by Bill Bird (Week 1)
 - Low-level Programming with C
 - Getting started with GenAI
 - Getting started with Bash
2. Bash (Weeks 2-3)
 - Bash commands
 - Shell scripting
3. C & Git (Weeks 3-5)
 - C and Bash
 - Pointers
 - Git commands
4. Python (Week 6+)
 - Python & Bash
 - PEP 8
 - Libraries
5. Answers to Selected Questions
6. Citations and References

Introduction to C: C Lectures by Bill Bird (Week 1)

Since the first lecture, I have progressively gone through the introductory C videos created and posted by Bill Bird (UVic legend). The majority of the content is easy to understand, with most functions/programs ranging from manipulating For Loops and Arrays to solve mathematical problems. As I have already taken several classes in Java, which is *very* similar to C, my understanding of these concepts is already reasonable. However, it is good in practice to test my learning and cover all of these topics again.

A particular example I finished after the second lecture was creating a program that computed the sum of n numbers, which can be found below:

In [109...

```
%%writefile program.c
#include <stdio.h>
```

```

int main() {
    int n, i;
    float num, sum = 0.0;

    // Get the number of elements from user
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    // Input n numbers and calculate their sum
    for(i = 0; i < n; i++) {
        printf("Enter number %d: ", i+1);
        scanf("%f", &num);
        sum += num;
    }

    // Display the result
    printf("Sum = %.2f\n", sum);

    return 0;
}

```

Overwriting program.c

```

In [54]: import os

# Compile the C Program
os.system('gcc program.c -o program')

# Run the compiled program
os.system('./program')

```

Out[54]: 1

I also worked on some other functions, including how to compute factorials, and other familiar low-level concepts from Calculus, like computing sums of a given array. Most of these functions are not entirely necessary to show in code, as they are easy to comprehend. Nonetheless, they gave me a more thorough understanding of how to use things like For Loops in C more effectively.

Low-level Programming with C

Most of this section was briefly covered above, with mentions of the functions I have worked on and the basics of what I've learned. As was mentioned, these different programs we are taught in Bill Bird's videos aren't really meant to be overly difficult. They are the definition of low-level, giving us a very basic yet thorough understanding of fundamental C concepts and how to create programs to complete simple tasks. In other words, they're just an efficient way to give us an understanding of the language before we begin work on assignments in the future. So far, I've found them to be quite useful!

Getting started with GenAI

During our first few classes, in between discussions about C intro, history, and operating systems, there have been several mentions of Generative AI and its usefulness in the programming space. We are highly encouraged to use these tools throughout our future careers, as well as during this semester (specifically for this class). While I already had a good understanding of Generative AIs such as Chat GPT and Julius (statistics engine), the discussions of this recently developing programming space were *very* interesting to learn about, in my opinion. I'm very interested to learn more.

During the lectures, the professor recommended the use of GenAI. One of our lectures provided us with a proper way to ask questions to AIs, in hopes that it could help us solve problems with our code. It also serves as a useful tool to learn Bash commands, as the AI provides descriptions of a command and how it is properly used when prompted.

Getting started with Bash

After our third lecture, we were given a brief look at several important bash commands that could become useful in the future. These commands were relatively basic, ranging from the echo command to various other commands for cycling through directories. Below are some of the commands I felt are interesting and useful:

```
In [58]: mkdir uvic
```

This Bash command creates a directory with the name "uvic", which can be cycled to/reached by using the **cd uvic** command.

```
In [67]: ls
```

Volume in drive C has no label.
Volume Serial Number is BEA4-B590

Directory of C:\Users\Arzia

```

03/16/2025  11:02 PM    <DIR>          .
10/25/2024  01:01 PM    <DIR>          ..
01/08/2025  10:44 AM    <DIR>          .anaconda
10/17/2024  01:39 PM    <DIR>          .cisco
01/08/2025  10:53 AM    <DIR>          .conda
01/08/2025  10:40 AM           146 .condarc
01/08/2025  10:39 AM    <DIR>          .continuum
03/16/2025  10:04 PM    <DIR>          .ipynb_checkpoints
01/08/2025  10:47 AM    <DIR>          .ipython
01/08/2025  10:44 AM    <DIR>          .jupyter
01/09/2025  02:46 PM    <DIR>          .lunarclient
10/17/2024  12:34 PM    <DIR>          .redhat
02/11/2025  12:42 AM    <DIR>          .ssh
01/21/2025  12:45 AM    <DIR>          .thumbnails
10/17/2024  12:49 PM    <DIR>          .vscode
08/26/2024  11:22 PM    <DIR>          3D Objects
08/26/2024  11:46 PM    <DIR>          ansel
10/25/2024  01:17 PM    <DIR>          Contacts
03/16/2025  04:42 PM    <DIR>          Desktop
02/11/2025  12:27 PM    <DIR>          Documents
03/16/2025  04:17 PM    <DIR>          Downloads
10/25/2024  01:17 PM    <DIR>          Favorites
10/25/2024  01:17 PM    <DIR>          Links
10/25/2024  01:17 PM    <DIR>          Music
01/08/2025  10:49 AM           1,380 my-notebook.ipynb
03/16/2025  01:10 PM    <DIR>          OneDrive
12/16/2024  11:47 PM    <DIR>          Pictures
03/16/2025  10:58 PM           472 program.c
01/30/2025  05:43 PM    <DIR>          Programming
10/25/2024  01:17 PM    <DIR>          Saved Games
10/25/2024  01:17 PM    <DIR>          Searches
03/16/2025  11:02 PM           8,781 TPP-P1-V01040408-Matthew-Goosney.ipynb
01/08/2025  10:44 AM           72 Untitled.ipynb
01/08/2025  10:44 AM           72 Untitled1.ipynb
03/16/2025  11:00 PM    <DIR>          uvic
03/16/2025  01:09 PM    <DIR>          Videos
           6 File(s)          10,923 bytes
          30 Dir(s)    7,829,229,568 bytes free

```

In [2]: `cd uvic`

C:\Users\Arzia\uvic

Above is the **ls** command, which provides us with important data, as well as information regarding our files.

Bash (Week 2-3)

During this lecture, we were given an introduction into how our Seng labs will be proctored, and what sort of general description we can expect to get from each encounter. We were also taught how to ssh into the UVic linux servers, which will allow us to access assignments and labs in the far future. The respective command for getting into the UVic server is found below:

```
In [74]: import os

# SSH into Linux servers
os.system('ssh matthewgoosney@ug1s.ece.uvic.ca')
```

Out[74]: 255

Bash commands

During one of our lectures in week three, we were given a list of the top 25 bash commands that could prove useful in future years of programming. Among these mentioned, one in particular really stood out to me: the **grep** search command. At it's most basic level, the command is used to search text for patterns (specified by the user), whether it has to do with the contents of a file or its name. I don't know why it stuck out, but I found it very interesting that it can be used to search inside files to find certain words, which seems very similar to the beloved ctrl + f command. I always found that command to be quite useful, so it's reassuring to know that bash has a similar command - albeit one with far more versatility and function.

As an aside, the focus on the top 25 commands gave me a bit of inspiration to do some digging myself, so I asked Chat GPT 4.0 the following question:

Prompt: What are the most useful bash commands other than cd and ls?

ChatGPT said: There are many useful Bash commands beyond cd (change directory) and ls (list files). Here are some of the most essential ones:

File and Directory Management

- pwd – Print the current working directory.
- mkdir dir_name – Create a new directory.
- rm file_name – Remove a file.
- rm -r dir_name – Remove a directory and its contents.
- mv old_name new_name – Rename or move a file or directory.
- cp source destination – Copy a file or directory (-r for directories).

File Viewing and Editing

- cat file_name – Display the contents of a file.
- tac file_name – Display a file in reverse order.

- `less file_name` – View a file interactively (scroll up/down).
- `head -n N file_name` – Show the first N lines of a file.
- `tail -n N file_name` – Show the last N lines of a file.
- `nano file_name` – Simple text editor.
- `vim file_name` – Advanced text editor.

Shell scripting

Along with the long list of bash commands and their various uses, we were also given a long list of bash scripts that serve a variety of purposes. They were decently interesting - however, I must admit I didn't learn much more than I'd already learned from asking Chat GPT questions. Of the few that did stick out to me, the `build3.sh.txt` script provided an example of how to compile code in a Linux environment.

```
#!/bin/bash gcc -c -Wall -std=c11 $1.c
```

It is relatively simplistic, and likely wouldn't come up anywhere on midterms, labs, or on assignments like more difficult bash scripts. However, I still found it useful regardless.

C & Git (Week 3-5)

At the end of our third week, we were given a sneak peak at our first assignment, which will include solving a variety of tasks that require us to take the data from a given CSV and YAML file respectively. We will then be required to paste specific functions into another CSV file as output. From a glance, this seems like a way to teach us how to properly parse data and handle large files. As of writing this, I have finished most of the assignment and completed most of the assigned tasks. In my opinion, this assignment was both entertaining and interesting. One thing I struggled with in particular was figuring out how to convert part of one dataset, the YAML file, which included a string column named "Extracurricular_Activities", and how to properly paste it into the output CSV file. It was extremely difficult, and took me over three hours to figure out on my own. However, I managed it. I solved it by taking the data from the YAML file and converting the string "yes/no" statement into a true or false value, then pasted it in the output file, and changed the truth value to either yes or no. There was likely a much, much simpler fix. However, I did whatever my brain could think of.

C and Bash

This week, we learned a lot more about C, Bash, and how to properly program in the Linux environment. We were also given a list of C filter programs. Of the few given, I took an interest in the **cipher.c** script, which implements a Caesar cipher encryption for lowercase letters. From my understanding, the program checks for lowercase letters, and once one is found, shifts it by two positions to the right in the alphabet. As an example, inputting "hello" would give you "jgnnq" as output.

Below is the code provided in our script file on the brightspace page:

```
In [107... %%writefile cipher.c
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>

#define ALPHABET_LENGTH 26
#define SHIFT_AMOUNT 2

int main(void) {
    int ch;

    while ((ch = getchar()) != EOF) {
        // Check if the character is a lowercase letter
        if (islower(ch)) {
            // Shift the letter two positions to the right
            ch = ((ch - 'a' + SHIFT_AMOUNT) % ALPHABET_LENGTH) + 'a';
        }
        putchar(ch);
    }

    return EXIT_SUCCESS;
} // main
```

Writing cipher.c

Pointers

Since before and after the first midterm, we have focused a lot on pointers and their purpose in C programming. From my understanding of the lectures and slides, a pointer in C is a sort of variable that stores the memory addresses of other variables. However, unlike regular variables like `int`, `string`, or `float` - which contain integers or characters - a pointer contains a reference to a **storage location**. The reason I believe this to be useful is because they allow for direct memory access and manipulation of data structures.

```
In [25]: %%writefile basic_pointer.c
#include <stdio.h>

int main() {

    // An integer variable
    int a = 10;

    // Create a pointer to integer (declaration)
    int * ptr;

    // Store the address of a inside pointer (initialization)
    ptr = &a;

    // Print the content of ptr
    printf("ptr = %p\n", ptr);
}
```

```
// Get the value pointed by ptr (dereferencing)
printf("*ptr = %d", *ptr);

return 0;
}
```

Writing basic_pointer.c

A good example of C Pointers and their application is the **malloc()** function, which allows us to allocate memory and later free said memory from use.

```
In [7]: %%writefile malloc.c

# Allocates memory
int* ptr = (int*)malloc(sizeof(int));
*ptr = 5;

# Deallocates memory
free(ptr);
```

Writing malloc.c

Of course, that isn't the only use that Pointers have in C. There are a wide range of uses that make these variables very useful in programming. For example, they can also be used for Call-By-Reference functions, allowing functions to modify variables directly:

```
In [24]: %%writefile modify.c

void modify(int* x) {
    *x = 20; // Modify the value stored at the address x points to
}

int main() {
    int num = 10;
    modify(&num);
    printf("%d", num); // Prints 20
}
```

Overwriting modify.c

Notes

- In call-by-references, a reference (address) to the original argument is passed to the function.
- Changes made to the parameter inside the function affect the value outside the function.
- Pointers are regularly used to implement call-by-references.

Git commands

In all honesty, I had a very rudimentary understanding of git commands prior to this course. I've participated in several hackathons since my first year, however, I never really understood how to actually push things to a repo - or how a repo and its server even worked, for that

matter. Rather, I always had another friend do it for me, as I was too lazy to learn it myself. However, since encountering this first assignment, I'm starting to get the hang of it. I've had several people ask me how to upload our assignments and labs since then, and been able to help and provide explanations from my own understanding. It's relatively straight-forward, and likely something that very few people would have difficulty learning. However, it is good information to have and quite useful in the long run, so I am glad that I've learned it.

```
git add -A git commit -m "Comments. (i.e.) Final Assignment Submission." git push
```

Python (Week 6+)

Python & Bash

The beginning of our journey with Python started with the basics of function calling and the use of the 'if **name** == **main**':" function, which from my understanding, works similarly to the main method concept in Java. In general, the simple calling block is used to determine whether a script is being run directly, or if it is being imported as a module. This allows a Python script to differentiate between execution modes, prevent certain code execution on imports, and just write reusable code in general. It funnels down the program's functions, ensuring that only the *necessary* parts are actually executed.

```
In [4]: #!/usr/bin/env python

def parabola():
    # Function contents for computing the vertex of a parabola
    return

if __name__ == "__main__":
    # This code will run if the script is executed directly
    print("Code execution. Vertex is x = 5.")
    parabola()
```

Code execution. Vertex is x = 5.

All things considered, Python seems like a much more *elegant* solution to programming than other languages like C or Java, which I feel have more complex methods to solving problems than something as streamlined as Python. Even in the beginning stages of this course, I knew that the actual power from Python comes from its extensive libraries, which honestly trivializes certain problems that we might encounter with other languages. A classic example of this is with the manipulation of CSV and YAML files, which require a lot more energy to tinker with if we were using another language. I discovered that personally while working on the first, second, and third assignments. Using C in general seemed simple enough, however, when we switched to Python for our second assignment, it was so much easier to grasp the concepts thanks to how versatile and VAST its libraries are. For our third assignment, I am certain that Python would've trivialized the use of Heaps and Linked-Lists - a problem that took me hours of tedium to solve.

How difficult were Assignment 2 and 3?

The leaps between Assignment 2 and 3, and the comparisons between Python and C respectively, are actually something that I wanted to explore a bit more in depth, namely in describing their difficulties. I want to begin with the second assignment, which relied on the use of Python to complete tasks that were largely similar to Assignment 1, albeit with an entirely different language. It was a much simpler process than working with C, as we were able to use libraries like **Pandas** to solve the problems we were given (writing and processing data from a given CSV file dataset). However, when we made the jump to the third assignment, which focused on the use of a Linked List and memory allocation to solve similar tasks - I feel that the difficulty level spikes drastically. While it is true that we were given files to help build our implementation, specifically "list.c" and "emalloc.c", there were more underlying issues that made writing the program difficult.

For instance, in Assignment 3 the CSV dataset itself seemed to have flaws - or at least from my reading of the data. Through every compilation cycle, I found that my output files were printing completely incorrect datasets, which the number of entries differing extremely from the test files we were given. As an example, my first test case had over 52 entries, whilst the second had 2 and the third had 11. I spent several hours debugging and combing through my code, in hopes that I'd find an error that caused this issue. In the end, I found nothing. In a final attempt, I spoke to two friends with knowledge of C programming, who also worked on the assignment separately. They came to the exact same issue - which leads me to believe that the dataset itself was flawed. I'm uncertain if this was an error on my part, or an error in the dataset, but regardless, the assignment itself was extremely difficult.

Working with Linked Lists is a very precise and intricate practice, especially when you add memory allocation into the mix. I constantly ran into errors about corrupted memory and mishandled nodes, which led me on a crusade of debugging and purging to find every misstep along the way. Unfortunately, I wasn't able to find a solution to the problem I was having, even after having refactored the majority of my code. Nonetheless, it was an important learning experience and certainly made me a better programmer by the end.

PEP 8

In short, PEP 8 is a guideline that shows the proper and most efficient style for writing clean, readable, and consistent Python code. It is the official style for Python, and just emphasizes the importance of keeping your code easy to read and maintain - which in and of itself is helpful when working with teams or when contributing to open-source projects. From my understanding, it is the Python equivalent to IEEE's plain language and readability rules, which ensure that, no matter what programmer reads the program, an understanding of its processes can be gathered.

1. Layout & Indentation

- Use four spaces per indentation level - do not use tabs.
- Limit all lines to a maximum of 79 characters for regular code, and 72 for docstrings.
- Use blank lines to separate functions, classes, and blocks of code inside functions.

```
In [ ]: # Correct:

# Aligned with opening delimiter.
foo = long_function_name(var_one, var_two,
                          var_three, var_four)

# Add 4 spaces (an extra level of indentation) to distinguish arguments from the re
def long_function_name(
    var_one, var_two, var_three,
    var_four):
    print(var_one)

# Hanging indents should add a level.
foo = long_function_name(
    var_one, var_two,
    var_three, var_four)
```

This example is taken from the PEP 8 website. Reference found below.

2. Imports

- All imports have an order that includes: Standard, Third-Party, Local application.
- All imports must be written line-by-line, meaning they cannot be included under one import.

```
In [ ]: # Correct:
import os
import sys

# Wrong
import sys, os
```

3. Naming Conventions

- Functions and variables use lower case letters with underscores between words.
- Class names used capitalized words (CamelCase).
- Constants use uppercase with underscores between words.
- Modules and packages use short, lowercase names with no underscores.
- Do not use "l", "I", or "O" as single character variable names.

4. White Space

- Never use spaces around the "=" sign when used to indicate a keyword argument.
- Use spaces around binary operators (a + b, x = 10).
- No space after the function name is in function calls (function(arg1, arg2)).

- Include spaces after the comma in function argument (function(a, b)).

5. Comments

- Block comments should be indented to the same level as the code they describe.
- Use complete sentences.
- Avoid obvious comments.
- Use inline comments sparingly.
- Use triple double quotations for docstrings to describe functions, classes, and modules.
- All functions and public classes should use docstrings.

There are many other recommendations in this guideline for programmers to follow. However, I feel that most of PEP 8 is already common knowledge and practice in our field, meaning that we all have an affluent understanding of how to properly write our code. As such, I feel that this basic summary hits most of the important points already! Nonetheless, I recognize that these guidelines are extremely important, and necessary if one truly wants to hone their skill in becoming a programmer who uses Python. Common practices like these are very easy to learn with practice, and should be one of the first things that an aspiring programmer does before working on complex projects.

Libraries

On that note, I feel it necessary to describe a few of the Python libraries that I myself have been tinkering with throughout this course. As I said in the above sections, the language itself has an almost endless list of libraries one can use to solve programming problems, varying from full-blown implementations of data structures to more niche and simplistic programs. It's an impressive scope, and likely a good reason as to why Python itself is considered the most versatile, effective, and widely-used programming language in the industry.

1. Pandas

Right from the get-go, I have to mention Pandas, as we used it first-hand on several occasions during the weeks of this class - whether for labs or for the second assignment. In general, Pandas is an extremely useful tool when it comes to data handling, manipulation, and analysis. I relied heavily on its use to solve Assignment 2. For instance, here is a small implementation that shows the initialization of a DataFrame, using the .describe command to show varied statistics:

```
In [46]: import pandas as pd

data = {'Name': ['Alice', 'Gremory', 'Charles'], 'Age': [25, 30, 35]}
df = pd.DataFrame(data)
```

```
print(df)
print(df.describe())
```

```
      Name  Age
0   Alice   25
1  Gremory   30
2  Charles   35

      Age
count  3.0
mean   30.0
std     5.0
min    25.0
25%    27.5
50%    30.0
75%    32.5
max    35.0
```

2. NumPy

Another great library that I found use for was NumPy, which is good for handling arrays and basic mathematical computations. As an example, I prompted ChatGPT to write a small implementation that computes the inverse of a matrix (2d array):

```
In [63]: import numpy as np

arr = np.array([1, 2, 3, 4, 5])
print(arr * 2)
matrix = np.array([[1, 2], [3, 4]])
print(np.linalg.inv(matrix))
```

```
[ 2  4  6  8 10]
[[-2.  1. ]
 [ 1.5 -0.5]]
```

I also used my own knowledge of NumPy to create another function that computes the vertex of a quadratic equation, that being $y = 2x^2 - 4x + 1$. Though it wasn't particularly difficult to make, I found it enjoyable, and thought it was a quality example of how useful this library can be:

```
In [68]: import numpy as np

def find_vertex(a, b, c):
    h = -b / (2 * a)
    k = a * h**2 + b * h + c
    return np.array([h, k])

vertex = find_vertex(2, -4, 1) # Values of the quadratic equation
print("The vertex is:", vertex)
```

```
The vertex is: [ 1. -1.]
```

3. Matplotlib

If it hasn't been obvious throughout this portfolio, I'm a bit of a math freak. Though my grades aren't incredibly impressive in these subjects, I have a deep interest in spheres of mathematics like Calculus (particularly in Integral Calculus and Infinite Series) and Matrix Algebra. For that reason, I've found myself researching a lot of libraries that centre around math and how to create programs for computation. Matplotlib is a good example of this, allowing for line plots and other graphs to visualize datasets. In my testing, there are definitely limitations - however, that doesn't take away from how useful it can be to just plot everything out to ease the understanding of concepts.

In the example below, I created a 3d donut (torus) using trigonometry - with the help of ChatGPT to make sure everything ran properly.

```
In [80]: import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Define the torus parameters
R = 3 # Major radius (distance from center to tube center)
r = 1 # Minor radius (tube radius)

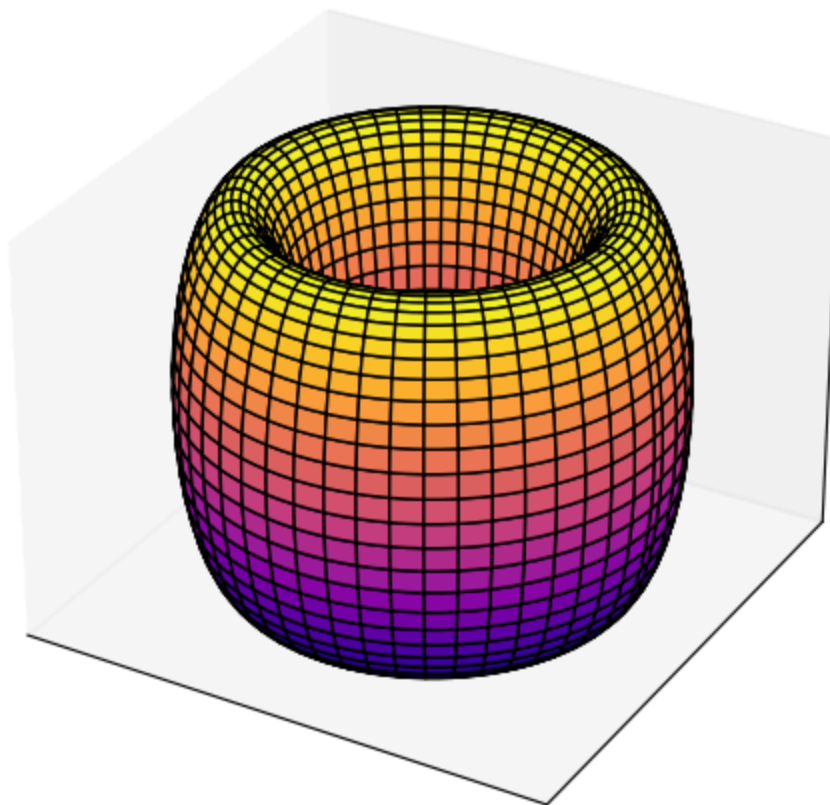
# Create a meshgrid for angles
theta = np.linspace(0, 2 * np.pi, 100) # Around the torus
phi = np.linspace(0, 2 * np.pi, 50) # Around the tube
theta, phi = np.meshgrid(theta, phi)

# Parametric equations for the torus
X = (R + r * np.cos(phi)) * np.cos(theta)
Y = (R + r * np.cos(phi)) * np.sin(theta)
Z = r * np.sin(phi)

# Create the 3D plot
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, Z, cmap='plasma', edgecolor='k')

# Remove axes for better visualization
ax.set_xticks([])
ax.set_yticks([])
ax.set_zticks([])

# Display the 3D torus
plt.show()
```



4. Turtle

On the topic of 3d creation, another library that I have found somewhat useful and interesting is Turtle, which allows the user to make shapes, patterns, and other graphics with ease. I haven't tinkered with it much yet, however, I have a feeling that it could be useful for our fourth and final assignment of this course. Unfortunately, Jupyter is unable to print the graphic as Turtle requires another tab to function (paste the graphic, meaning), but the concept is easy enough to understand without printing code.

```
In [ ]: import turtle

# Set up the turtle
t = turtle.Turtle()

# Draw a square
for _ in range(4):
    t.forward(100)
    t.left(90)

turtle.done()
```

5. PyBrain

Finally, the last library that I feel like mentioning is PyBrain, which provides a lot of useful tools for creating artificial intelligence and machine learning. It is relatively complex, especially for someone without knowledge on the aforementioned subjects. However, it is also modular, and is really effective at building machine-learning models. I haven't dedicated much time to using it, however, I intend to do so in the coming months (summer vacation to work on projects).

Answers to Selected Questions

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

- Git is used as a version control system thanks to its abilities to track file changes - allowing programmers to revert to previous versions if need be - create separate branches from projects to work from independently, and remerge at later dates; this can be done thanks to each dev having a local repository with full project history. It allows for amazing collaboration, as multiple devs can work on the same files without getting conflicts - so long as everyone is vigilante. Overall, Git is a very effective tool in the programming space, one that I dare say is a necessity in jobs and projects where programmers must work together.

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

- By far, the most popular programming languages are likely Python, JavaScript, and Java. Python itself is likely at the top, as it is extremely easy to read and write, with various uses in AI and web development (and countless other areas). Furthermore, Python's extensive libraries make it extremely easy to actually code, with countless functions that can do pretty much anything if you look hard enough. A good example of these libraries is pandas, which we have learned firsthand in our classes and labs. Another popular language is JavaScript, which is likely the king of web development. To my knowledge, most, if not all browsers (Google, Microsoft Edge, etc) use JavaScript for frontend development. It is also connected to various frameworks, such as Node.js and React. I myself have used these frameworks before, and they are quite interesting. Finally, I'd say that Java is another one of the most popular languages. It is used in a lot of development spaces, including Android apps, other applications, and backend systems. As a fun, very well-known fact, the game Minecraft is designed using Java, which definitely adds a bit to its popularity as the game has an extensive modding community. There are several others, including C, C++, C#, and SQL, however, I feel that these languages are not as popular due to their more niche use-cases.

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

- Before this course, I had never actually touched the C programming language. Throughout my two years of school, I have only ever learned languages such as Java

and Python - the former I am much more versed in. In regards to its prevalence, I still believe that C is a foundational programming language that will continue to see use for decades to come. All of our major operating systems, including Linux, Windows, and macOS, use C for its computational abilities and memory management. In that aspect, it is the king. I'm certain that it also has some use in game development and graphics engines, such as Unreal Engine (I'd heard about this when I first found out about Unreal Engine a few years ago). Outside of these aspects, I am certain that C is used in numerous other places, which really goes to show just how useful and impressively versatile C is.

4. How challenging was learning C for you?

- Despite me having almost zero knowledge of C prior to this course, programming with it for our assignments and labs was actually quite simpler than I expected. On the surface, it seems very similar to how Java is oriented (I understand that Java was based off of C), which made it much easier to create working functions and solve debugging issues. I thought I'd need to do several weeks of catch-up to understand it all to the same degree as Software Engineering students, who learn the language in their first year. However, I was surprised to learn that I needed almost no time at all to catch up.

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

- In short, SENG 265 has been an incredibly useful tool for propelling my resume. Before this class, I was severely lacking in confidence for my abilities as a programmer, to the point where I thought that this might not have been the right career path for me. However, after having learned languages like C and Python, and actually built projects that gave us useful learning experiences (like how to manipulate common data files, which we will most definitely see in co-ops), my confidence has grown. It taught me that I am not a bad programmer and simply haven't done enough to propel my knowledge. Furthermore, it showed me that if I put the time and energy into seeing a project done, it will get done.

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

- This question can be interpreted in several ways. However, I'll do my best to answer it. I first experienced GenAI in my first year of university, when a few videos on my youtube recommended began popping up. I never really used it until this last year, where its proven very useful in solving debugging issues that I just can't figure out, or helping with writing a function that I can't wrap my head around. I'm good at not using it for assignments, as I feel it is important as a student to learn the material, rather than using AI to figure out every problem. That being said, AI is also wrong quite often.
- As a learning tool specifically, I found GenAI tools to be extremely useful and effective. While it is true that they get things wrong sometimes when it comes to writing intricate

programs, they are very good at describing concepts in an easy and concise manner, whilst also providing sufficient examples to really show you how something is done. I often use it to explain programming concepts I can't quite understand, like the intricacies of machine learning, or if there is a problem with an implementation that I just can't see. In other words, it is a good second opinion, one that you can turn to if you're in a rough spot and can't see a way to solve a problem.

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

- A few of my insights and epiphanies can be found in some of the above sections of this portfolio. However, to be thorough, I'll briefly mention it here. So far, the most significant epiphany I've had is the ease and usefulness of Git, which before this course I never had any understanding of.
- After the conclusion of the third assignment, I experienced an epiphany after over six hours of continuous debugging: the CSV dataset was possibly incorrect. Myself and several others were experiencing issues with the data, wherein we were getting far more entries than the provided tester files gave us.

Citations and References

1. Information about C Pointers, GeeksforGeeks (2024) <https://www.geeksforgeeks.org/c-pointers/>
2. Information about PEP 8, Python (2001) <https://peps.python.org/pep-0008/>
3. SENG 265 Slides and Lecture information, UVic (2025)
4. Chat GPT for prompts and small programming task questions, OpenAI (2025) <https://chatgpt.com/>