Unit 7 – Codio:

1. Exploring a simple Python shell

In this session, you will create a command shell in Python and then run it and answer questions about it. You can use the **Jupyter Notebook space in Codio** for your work.

Review the blogs at Praka (2018) and Szabo (n.d.) and then create a CLI/ shell that implements the following:

- When you enter the command LIST, it lists the contents of the current directory
- The ADD command will add the following two numbers together and provide the result
- The HELP command provides a list of commands available
- The EXIT command exits the shell

Add suitable comments to your code and add the program to your e-portfolio. **Be prepared to demonstrate it in the seminar session next week.**

Run the shell you have created, try a few commands and then answer the questions below. Be prepared to discuss your answers in the seminar.

- What are the two main security vulnerabilities with your shell?
- What is one recommendation you would make to increase the security of the shell?
- Add a section to your e-portfolio that provides a (pseudo)code example of changes you would make to the shell to improve its security.

Remember to also record your results, ideas and team discussions in your e-portfolio.

Learning Outcomes

- Identify and manage security risks as part of a software development project.
- Critically analyse development problems and determine appropriate methodologies, tools and techniques (including program design and development) to solve them.
- Design and develop/adapt computer programs and to produce a solution that meets the design brief and critically evaluate solutions that are produced.

Python is a powerful programming language useful for workflow automation and scripting. It includes several useful tools from the Python Standard Library. System administrators frequently use shell commands and can run using Python. Python can complete various traditionally carried-out jobs using batch or bash files (Janakiev, 2019).

Below is an example of a command shell in Python to execute specific commands, such as LIST, ADD, HELP, and EXIT, based on the ideas of Praka (2018) and (Arbitrio, N.D):

```
"""
This is a CLI/shell written in Python
that implements the following commands:
LIST: Lists the contents of the current directory.
ADD: Adds two numbers together and provides the result.
HELP: Provides a list of commands available.
EXIT: Exits the shell.
"""

import cmd
import os

class MyPrompt(cmd.Cmd):
    """
    Display a welcome message when
    the application is launched
    """
    prompt = "myshell> "
    intro = "Welcome to MY shell! Type help to list commands"

    def __init__(self):
        super().__init__()
        self.current_dir = os.getcwd()

    def do_LIST(self, args):
        """List the contents of the current directory."""
        print(os.listdir(self.current_dir))

    def do_ADD(self, args):
        """Add two numbers together and provide the result."""
        numbers = args.split(" ")
        num1 = input('Enter first number: ')
        num2 = input('Enter second number: ')

        # Add two numbers
        sum = float(num1) + float(num2)

        # Display the sum
        print('The result is {2}'.format(num1, num2, sum))
    def do_HELP(self, args):
        """Provide a list of available commands."""
        print("Available commands:")
        print("LIST: List the contents of the current directory.")
        print("ADD: Add two numbers together and provide the result.")
        print("HELP: Provide a list of available commands.")
        print("EXIT: Exit the shell.")

    def do_EXIT(self, args):
        """Exit the shell."""
        print("Goodbye!")
        exit()


if __name__ == "__main__":
    prompt = MyPrompt()
    prompt.cmdloop()
```

Output:

```
C:\Users\hcham\anaconda3\envs\pythonProject-SSD\python.exe
C:\Users\hcham\PycharmProjects\pythonProject-SSD\shell-3.py

Welcome to MY shell! Type help to list commands
myshell> help

Documented commands (type help <topic>):
========================================
ADD  EXIT  HELP  LIST  help

myshell> help ADD
Add two numbers together and provide the result.
myshell> ADD
Enter the first number: 1
Enter the second number: 1
The result is 2.0
myshell> help LIST
List the contents of the current directory.
myshell> LIST
['.idea', 'hanoi_pygame.py', 'recursion.py', 'regex.py', 'shell-1.py', 'shell-2.py', 'shell-3.py', 'shell.py']
myshell> help HELP
Provide a list of available commands.
myshell> HELP
Available commands:
LIST: List the contents of the current directory.
ADD: Add two numbers together and provide the result.
HELP: Provide a list of available commands.
EXIT: Exit the shell.
myshell> help EXIT
Exit the shell.
myshell> EXIT
Goodbye!

Process finished with exit code 0
```

1. **What are the two main security vulnerabilities with your shell?**

Application security can be compromised by a harmful technique called command injection, which involves injecting malicious data into the server and executing it. Even languages known for their secure design, such as Python, can be vulnerable to this attack. Using unsanitised user input creates a vulnerability that can be exploited in this manner (StackHawk, 2021).

My code lacks input validation and has two security vulnerabilities: command injection and input validation.

2. **What is one recommendation you would make to increase the security of the shell?**

Zhong (N.D.) states that command injection is an attack whereby an attacker can execute arbitrary commands on a host operating system through a vulnerable application. This occurs when an application fails to validate user inputs and passes them on to a shell. The attacker's commands are then executed using the same privileges as the application. Unlike code injection, command injection doesn't involve injecting new code but exploits the application's existing functionality.

To increase the security of my code, I would change the input validation, which is crucial for safety, preventing unintended input, modified control flow, and arbitrary code execution. It involves filtering, encoding/escaping, and checking data properties such as well-formedness, consistency, and adherence to rules. As CWE (2009) states, developers must differentiate input validation from output escaping to avoid vulnerabilities caused by property derivation errors.

3. **Add a section to your e-portfolio that provides a (pseudo)code example of changes you would make to the shell to improve its security.**

As Zhu (2023) suggested, a way to improve app security is by validating and sanitising user inputs before execution. This approach can prevent malicious attacks such as command injection.

Below is an example of the pseudo(code) changes that can be added to improve security:

```
def do_ADD(self, args):
   # Input validation
   try:
      num1 = float(args.split(" ")[0])
      num2 = float(args.split(" ")[1])
   except ValueError:
      print("Invalid input.")
      return

   # User input sanitiser
   num1 = str(num1).strip()
   num2 = str(num2).strip()

   # Execute the command.
   result = num1 + num2
   print(result)
```

4. **Developing an API for a Distributed Environment**

In this session, you will create a RESTful API which can be used to create and delete user records. Responses to the questions should be recorded in your e-portfolio.

You are advised to use these techniques to create an API for your team's submission in Unit 11 and be prepared to demonstrate it during next week's seminar (Unit 10). Remember that you can arrange a session with the tutor during office hours for more support, if required.

Using the **Jupyter Notebook workspace**, create a file named api.py and copy the following code into it (a copy is provided for upload to Codio/GitHub): You can **install**

Jupyter Notebook on your local machine following these instructions or via the University of Essex Software Hub.

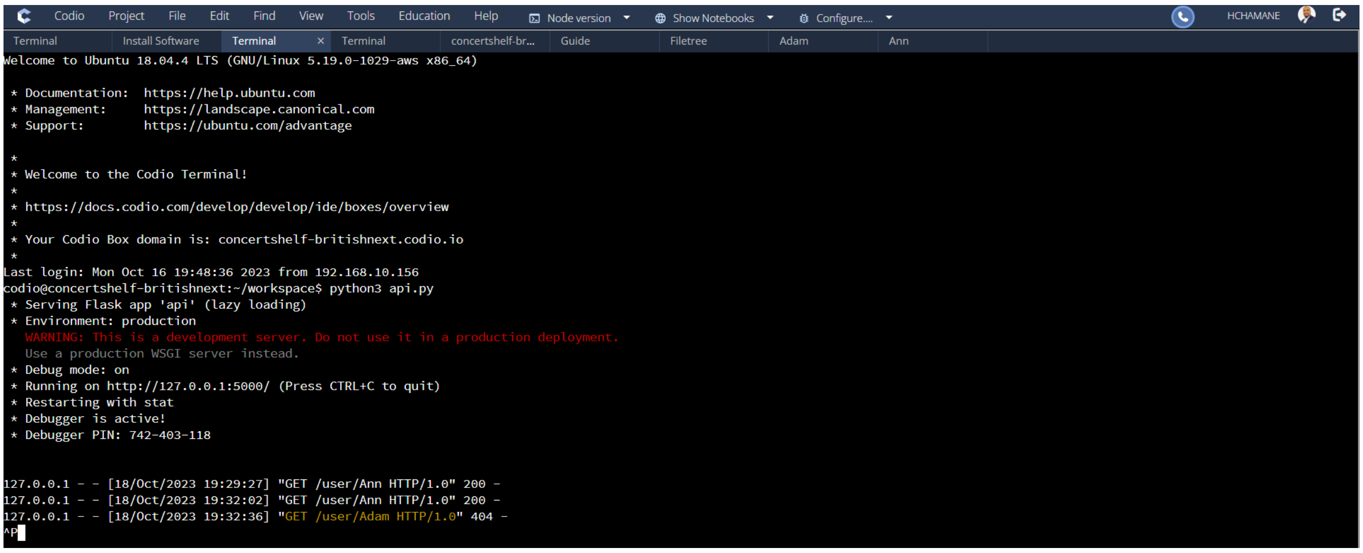
#source of code: Codeburst

```
from flask import Flask
from flask_restful import Api, Resource, reqparse

app = Flask(__name__)
api = Api(app)

users = [
    {
        "name": "James",
        "age": 30,
        "occupation": "Network Engineer"
    },
    {
        "name": "Ann",
        "age": 32,
        "occupation": "Doctor"
    },
    {
        "name": "Jason",
        "age": 22,
        "occupation": "Web Developer"
    }
]

class User(Resource):
    def get(self, name):
        for user in users:
            if(name == user["name"]):
                return user, 200
        return "User not found", 404

    def post(self, name):
        parser = reqparse.RequestParser()
        parser.add_argument("age")
        parser.add_argument("occupation")
        args = parser.parse_args()

        for user in users:
            if(name == user["name"]):
                return "User with name {} already exists".format(name), 400

        user = {
            "name": name,
            "age": args["age"],
            "occupation": args["occupation"]
        }
        users.append(user)
        return user, 201

    def put(self, name):
        parser = reqparse.RequestParser()
        parser.add_argument("age")
        parser.add_argument("occupation")
        args = parser.parse_args()

        for user in users:
            if(name == user["name"]):
                user["age"] = args["age"]
                user["occupation"] = args["occupation"]
                return user, 200

        user = {
            "name": name,
            "age": args["age"],
            "occupation": args["occupation"]
        }
        users.append(user)
        return user, 201

    def delete(self, name):
        global users
        users = [user for user in users if user["name"] != name]
        return "{} is deleted.".format(name), 200

api.add_resource(User, "/user/<string:name>")

app.run(debug=True)
```

Question 1

Run the API.py code. Take a screenshot of the terminal output. What command did you use to compile and run the code?

To compile and run the code, I used *python3 api.py,* as shown in the image below:



```
Welcome to Ubuntu 18.04.4 LTS (GNU/Linux 5.19.0-1029-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 *
 * Welcome to the Codio Terminal!
 *
 * https://docs.codio.com/develop/develop/ide/boxes/overview
 *
 * Your Codio Box domain is: concertshelf-britishnext.codio.io
 *
Last login: Mon Oct 16 19:48:36 2023 from 192.168.10.156
codio@concertshelf-britishnext:~/workspace$ python3 api.py
 * Serving Flask app 'api' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 742-403-118


127.0.0.1 - - [18/Oct/2023 19:29:27] "GET /user/Ann HTTP/1.0" 200 -
127.0.0.1 - - [18/Oct/2023 19:32:02] "GET /user/Ann HTTP/1.0" 200 -
127.0.0.1 - - [18/Oct/2023 19:32:36] "GET /user/Adam HTTP/1.0" 404 -
^P
```

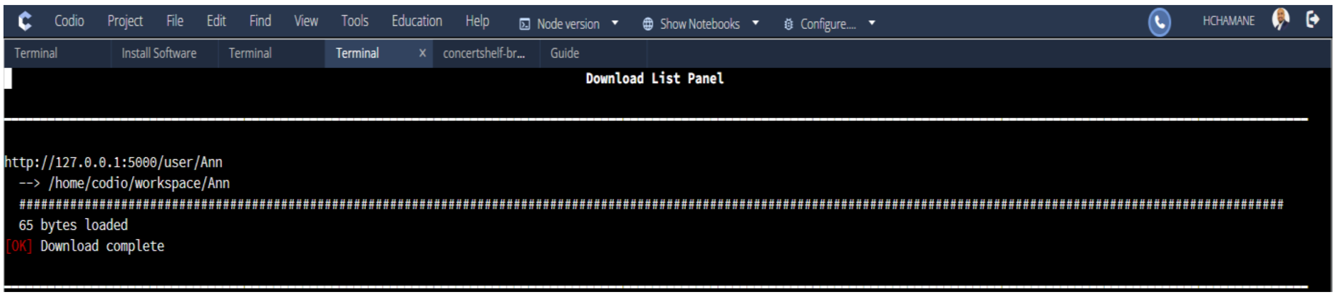
Question 2

[Run the following command](#) at the terminal prompt:

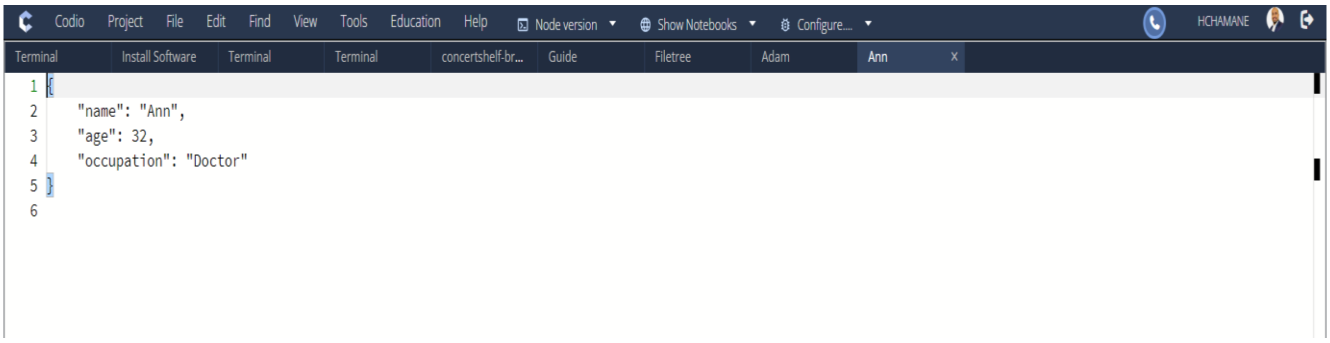
w3m http://127.0.0.1:5000/user/Ann

What happens when this command is run, and why?

When I execute the command w3m, it downloads the content of the user Ann, allowing me to browse the content in text format, as shown in the images below:



```
                                   Download List Panel
--------------------------------------------------------------------------------
http://127.0.0.1:5000/user/Ann
  --> /home/codio/workspace/Ann
  ##############################################################################
  65 bytes loaded
[OK] Download complete
--------------------------------------------------------------------------------
```



```
{
    "name": "Ann",
    "age": 32,
    "occupation": "Doctor"
}
```

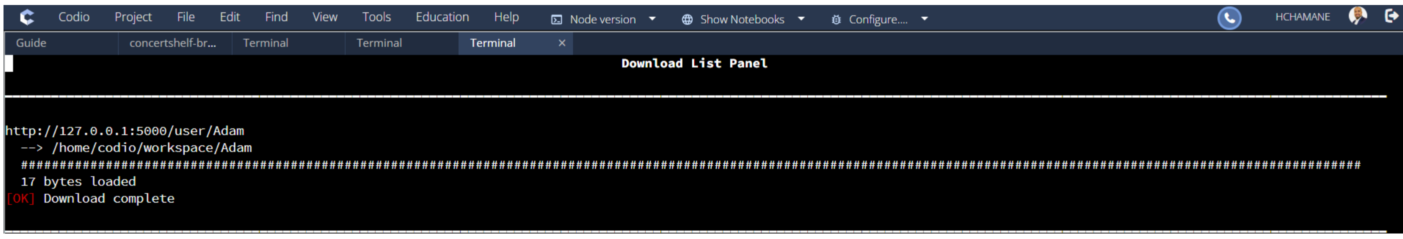
Question 3

[Run the following command](#) at the terminal prompt:

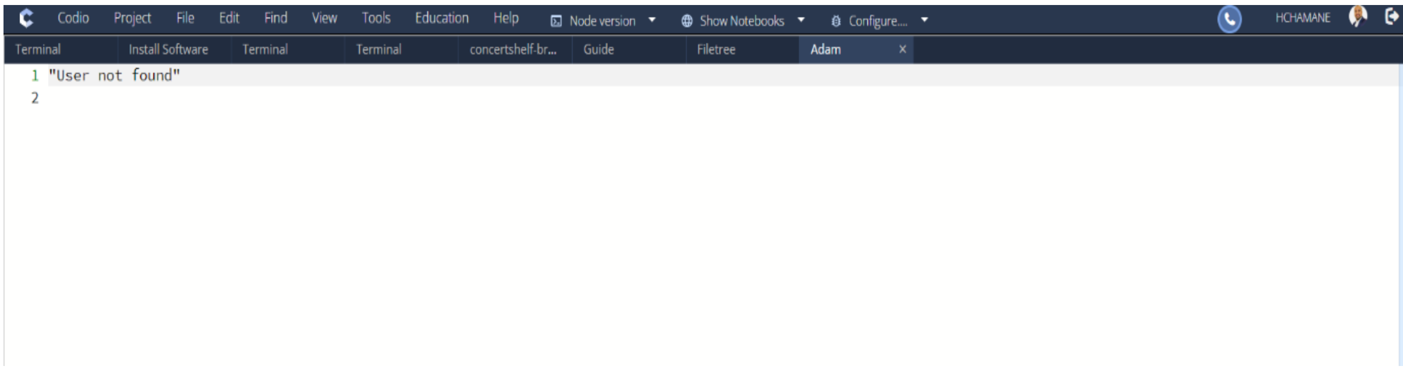
w3m http://127.0.0.1:5000/user/Adam

What happens when this command is run, and why?

When I tried to browse the content of user Adam using the w3m command, the result was "User not found" because the user Adam does not exist, as shown in the images below:



```
                                        Download List Panel

http://127.0.0.1:5000/user/Adam
  --> /home/codio/workspace/Adam
  ##########################################################################################################################################################################
  17 bytes loaded
[OK] Download complete
```



```
"User not found"
```

Question 4

What capability is achieved by the flask library?

According to Smyth (2018), flask maps HTTP requests to Python functions. When we connect to the Flask server, it checks if there is a match between the path provided and a defined function. Flask displays the returned result in the browser. In this case, it returns an HTML markup for a home page welcoming visitors to the site hosting our API.

Flask is a lightweight and flexible microweb framework for Python, allowing developers to quickly build web apps and APIs (Dyouri, 2020).

In addition, as highlighted by Murugan (2023), Flask boasts a built-in development server, support for unit testing and RESTful request dispatching, secure cookies, and compliance with WSGI (Web Server Gateway Interface). Its flexibility and extensibility have made it a favoured option. Still, it has a few drawbacks, such as a challenging learning curve, limited functionality, lack of built-in database integration, and potential security vulnerabilities.

Architecture Evolution Activity

Based on your reading this week, could you write a section that might be appended to this paper, Salah et al. 2016, presenting the next phase of evolution history, from microservices to the technologies commonly used today?

My findings:

Throughout history, humans have consistently innovated new forms of communication - from smoke signals to emails. The invention of the electric telegraph in 1831 represented a significant breakthrough in communication technology, but now digital methods have taken over. Today, we tend to avoid handwriting letters or making phone calls, instead opting to communicate via email, FaceTime, Zoom, or Google Hangouts. The amount of progress that has been made in the realm of communication is truly remarkable (Rogers, 2019).

Today's businesses face the challenge of meeting customer demands while navigating complex technologies. Up to 45% of employee activities can be automated, leading to a shift in the definition of "services." Top performance can be achieved through mergers, investments, resource allocation, and productivity improvements. A next-gen model must capitalise on digital services to support speed, agility, efficiency, and precision (McKinsey, 2017).

Salah et al. (2016) state that communication networks have led to the development of distributed software systems offering real-time services to meet growing user demands. Different paradigms like Client-Server, SOA, and Microservices have emerged to structure these applications based on their unique needs, offering efficient ways to handle requests, integrate systems, and break down complex applications for greater flexibility and faster deployment.

Hence, microservices offer benefits like software reuse, improved availability, and continuous delivery (Assunção et al. 2023). However, managing knowledge about their dependencies can be challenging, leading to unnecessary costs and design flaws. Microservices typically co-evolve with each other, driven by various factors such as business requirements, technical dependencies, and organisational changes.

References:

Janakiev, N. (2019). *How to Execute Shell Commands with Python*. [online] Parametric Thoughts. Available at: https://janakiev.com/blog/python-shell-commands/.

Praka, D. (2018). *Write a shell in Python — Danish Prakash*. [online] Available at: https://danishpraka.sh/posts/shell-in-python/.

Arbitrio, E. (N.D). *Create your custom interactive shell with python | Ernesto Arbitrio*. [online] Available at: https://ernesto.arbitrio.fyi/posts/create-yout-own-interactive-shell-with-python/.

StackHawk (2021). *Command Injection in Python: Examples and Prevention*. [online] Available at: https://www.stackhawk.com/blog/command-injection-python/

Zhong, W. (N.D.). *Command Injection* | *OWASP*. [online] owasp.org. Available at: https://owasp.org/www-community/attacks/Command_Injection.

CWE (2009). *CWE - CWE-20: Improper Input Validation (3.4.1)*. [online] Available at: https://cwe.mitre.org/data/definitions/20.html.

Zhu, M (2023). *User input sanitization and validation* | *TinyMCE*. [online] Available at: https://www.tiny.cloud/blog/input-sanitization/

PHO (2023). *Making a simple calculator python*. [online] Available at: https://stackoverflow.com/questions/75677334/making-a-simple-calculator-python

Smyth, P. (2018). Creating Web APIs with Python and Flask. *Programming Historian*. [online] Available at: https://programminghistorian.org/en/lessons/creating-apis-with-python-and-flask.

Dyouri, A. (2020). *How To Make a Web Application Using Flask in Python 3*. [online] DigitalOcean. Available at: https://www.digitalocean.com/community/tutorials/how-to-make-a-web-application-using-flask-in-python-3.

Murugan, S. (2023). *Top Python REST API Frameworks in 2023*. [online] Available at: https://www.browserstack.com/guide/top-python-rest-api-frameworks.

Rogers, S. (2019). *The Role Of Technology In The Evolution Of Communication*. [online] Forbes. Available at: https://www.forbes.com/sites/solrogers/2019/10/15/the-role-of-technology-in-the-evolution-of-communication/.

McKinsey (2017). *McKinsey on Digital Services Introducing the next-generation operating model*. [online] Available at: https://www.mckinsey.com/~/media/McKinsey/Business%20Functions/McKinsey%20Digital/Our%20Insights/Introducing%20the%20next-generation%20operating%20model/Introducing-the-next-gen-operating-model.ashx.

Salah, T., Jamal Zemerly, M., Chan Yeob Yeun, Al-Qutayri, M. & Al-Hammadi, Y. (2016). *The evolution of distributed systems towards microservices architecture*. [online] IEEE Xplore. doi:https://doi.org/10.1109/ICITST.2016.7856721.

Assunção, W.K.G., Krüger, J., Mosser, S. & Selaoui, S. (2023). How do microservices evolve? An empirical analysis of changes in open-source microservice repositories. *Journal of Systems and Software*, [online] 204, p.111788. doi:https://doi.org/10.1016/j.jss.2023.111788.

Unit 7 – Reflection:

Throughout unit 7 studies, I have gained a wealth of knowledge regarding operating systems' various types and functions and deepened my understanding of different processes. In particular, I have learned critical approaches that can be employed to enhance the security of operating systems, which is of utmost importance to me as an infrastructure engineer who has extensive experience working with operating systems. Furthermore, I have explored the differences between various virtualisation approaches, gaining valuable insights into their practical applications. I have had the opportunity to engage in Codio activities, where I studied the practical applications of libraries in real-world scenarios, including a challenging Python shell.

One of the week's highlights was the lecture on operating system security, which provided an excellent recapitulation of the operating system's history and evolution. It also touched on the various legal, ethical, social, and professional considerations surrounding the development of operating systems. Throughout my experience working on my shell code this week, I encountered some challenges that proved quite daunting, but I was determined to push through and eventually overcame those obstacles. This week's learning proved extremely valuable, as I gained essential knowledge and skills that will undoubtedly enhance my work in the future. Specifically, I now better understand concepts such as creating a command shell in Python, which will be crucial in my future coding endeavours. Overall, while it was a challenging week, I am grateful for the immense growth I experienced.