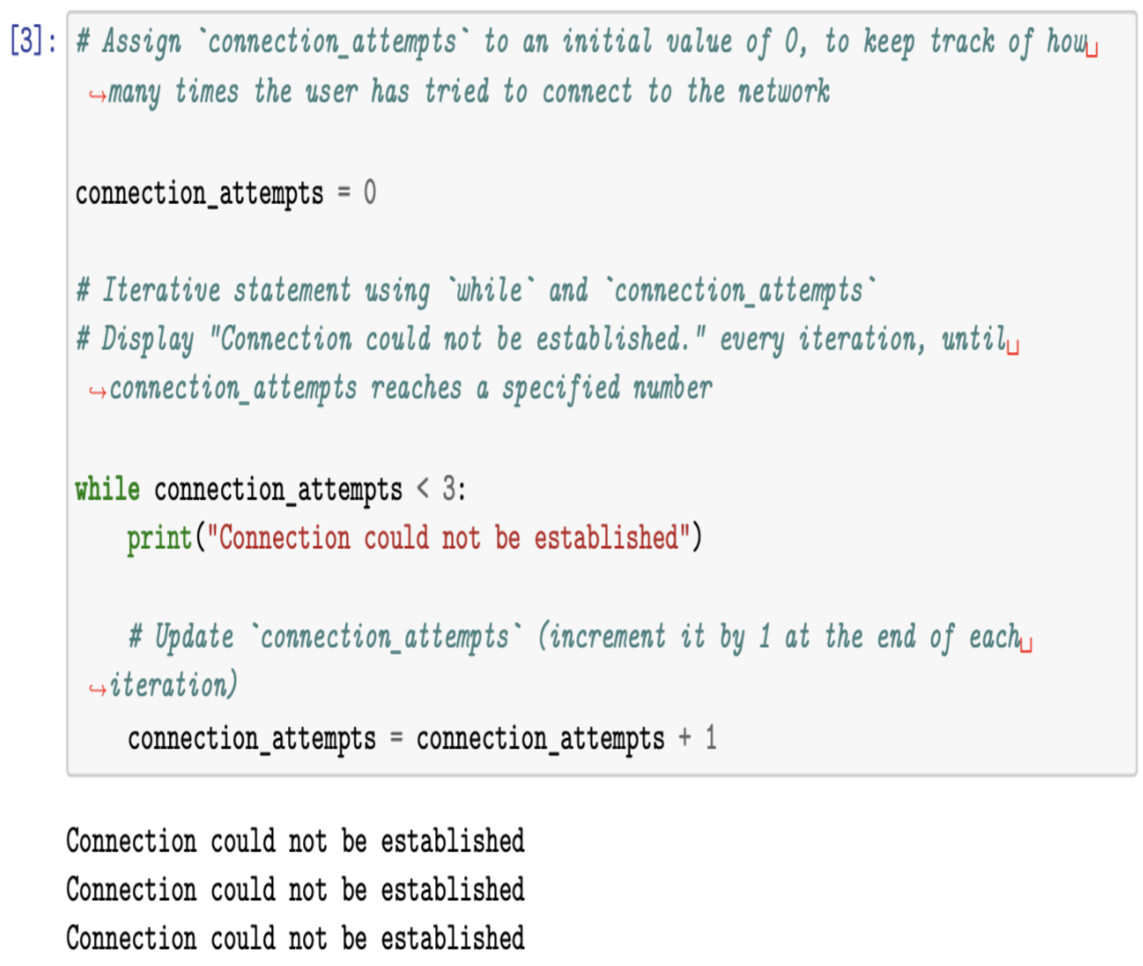
Python Control Flow and Iteration Project

In this project, I practiced fundamental Python programming concepts focused on **control flow**, **loops**, and **conditional logic** — all essential skills for cybersecurity automation, data analysis, and network scripting. The tasks revolved around using while loops and if-else statements to solve real-world inspired problems.

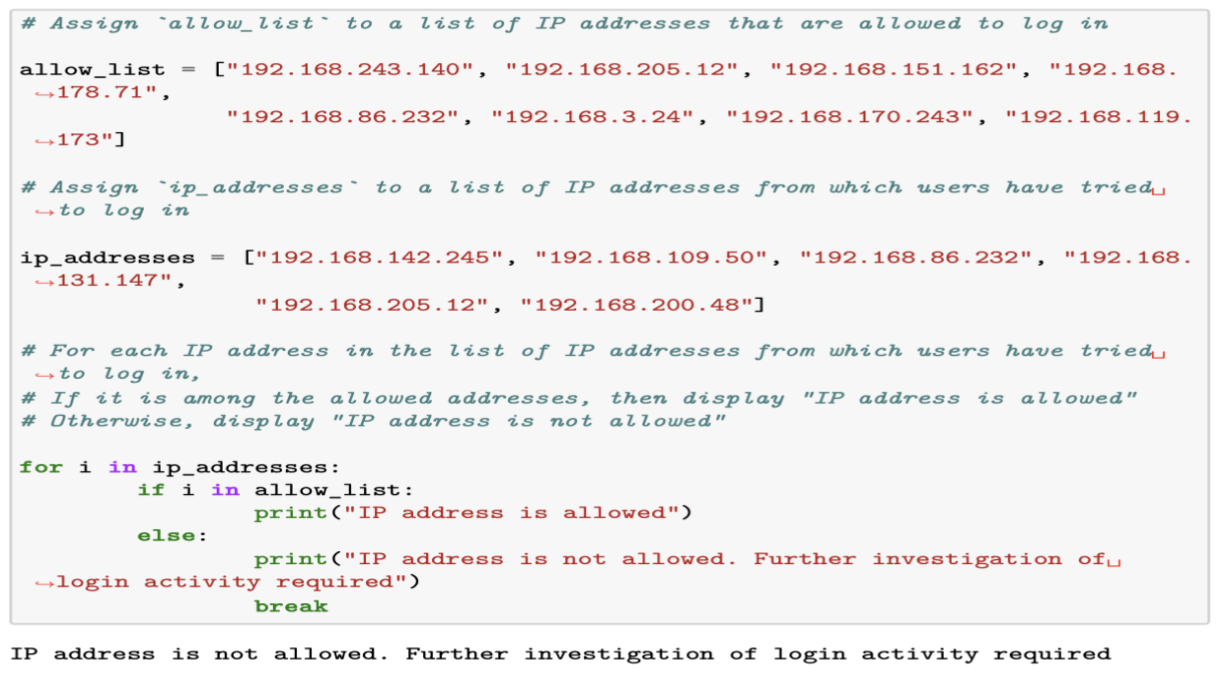
The first task involved creating a program to **track network connection attempts**. I initialized a variable connection_attempts to 0 and used a while loop to repeatedly display a message, “Connection could not be established,” until the number of attempts reached a set limit. This simulated a common network diagnostic scenario where repeated attempts are logged when a connection fails.



```
[3]: # Assign `connection_attempts` to an initial value of 0, to keep track of how␣
      ↪many times the user has tried to connect to the network

     connection_attempts = 0

     # Iterative statement using `while` and `connection_attempts`
     # Display "Connection could not be established." every iteration, until␣
      ↪connection_attempts reaches a specified number

     while connection_attempts < 3:
         print("Connection could not be established")

         # Update `connection_attempts` (increment it by 1 at the end of each␣
      ↪iteration)
         connection_attempts = connection_attempts + 1
```

```
Connection could not be established
Connection could not be established
Connection could not be established
```

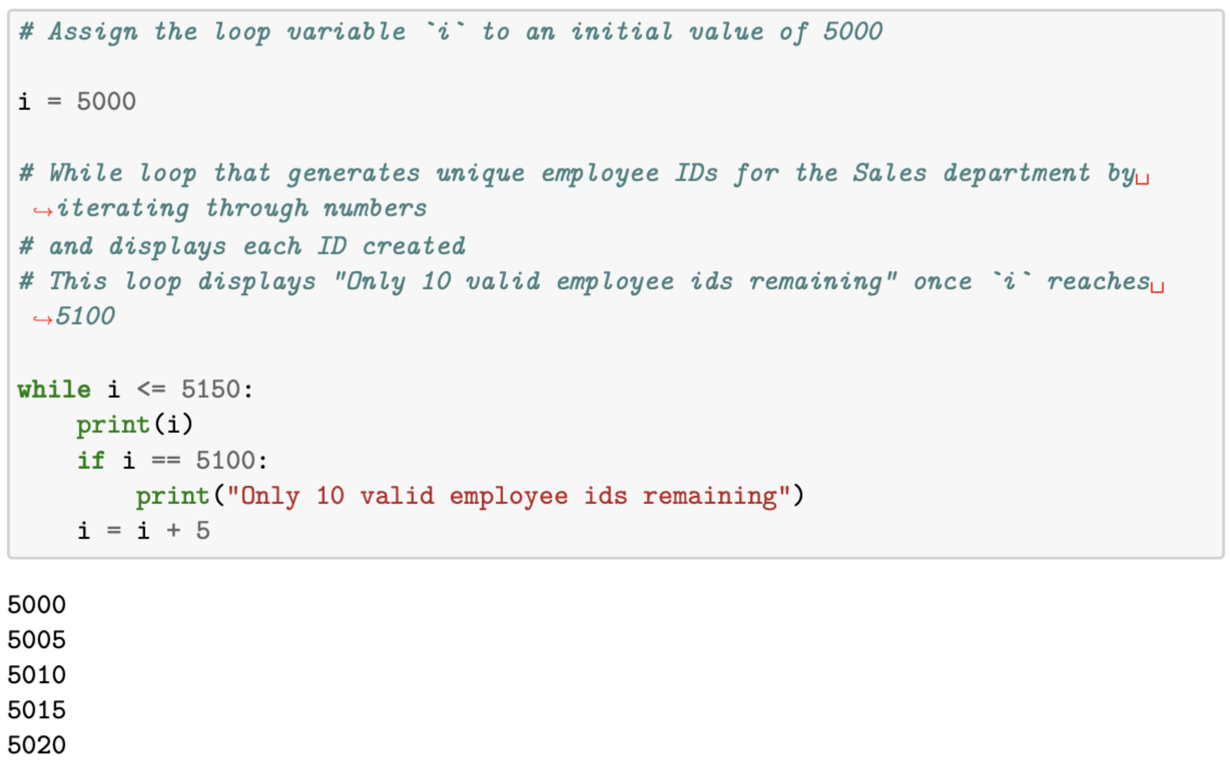
Next, I implemented a script to **check and validate IP addresses** against an allowlist. By defining two lists — one containing authorized IP addresses (allow_list) and another with incoming connection attempts (ip_addresses) — I used a for loop and conditional statements to verify access. If an IP address was authorized, the program printed a confirmation message. If not, it displayed a warning message indicating that further investigation was required. This mirrors the type of logic used in security monitoring systems to flag suspicious login attempts.



```
# Assign `allow_list` to a list of IP addresses that are allowed to log in

allow_list = ["192.168.243.140", "192.168.205.12", "192.168.151.162", "192.168.
 ↪178.71",
              "192.168.86.232", "192.168.3.24", "192.168.170.243", "192.168.119.
 ↪173"]

# Assign `ip_addresses` to a list of IP addresses from which users have tried␣
 ↪to log in

ip_addresses = ["192.168.142.245", "192.168.109.50", "192.168.86.232", "192.168.
 ↪131.147",
                "192.168.205.12", "192.168.200.48"]

# For each IP address in the list of IP addresses from which users have tried␣
 ↪to log in,
# If it is among the allowed addresses, then display "IP address is allowed"
# Otherwise, display "IP address is not allowed"

for i in ip_addresses:
        if i in allow_list:
                print("IP address is allowed")
        else:
                print("IP address is not allowed. Further investigation of␣
 ↪login activity required")
                break
```

```
IP address is not allowed. Further investigation of login activity required
```

Finally, I created a loop that **generated unique employee IDs** for a department. Starting with an initial value of 5000, the script iterated through a sequence of numbers and displayed each generated ID. Once it reached a specific threshold (5100), it printed a special message: “Only 10 valid employee IDs remaining.” This exercise demonstrated how iteration can be used for automated ID generation and resource tracking in enterprise applications.



```
# Assign the loop variable `i` to an initial value of 5000

i = 5000

# While loop that generates unique employee IDs for the Sales department by␣
 ↪iterating through numbers
# and displays each ID created
# This loop displays "Only 10 valid employee ids remaining" once `i` reaches␣
 ↪5100

while i <= 5150:
    print(i)
    if i == 5100:
        print("Only 10 valid employee ids remaining")
    i = i + 5
```

```
5000
5005
5010
5015
5020
```

Overall, this project strengthened my understanding of how Python's iterative structures and conditionals can be applied in practical scenarios — from network connection monitoring and access control to automated data generation. These skills are fundamental to writing scripts that support cybersecurity workflows and IT automation.