🔐 Project: Automating Access Control & Employee ID Provisioning Using Python Loops and Conditional Logic

---

🧨 Problem Context: Simulating Real-World Network Access Monitoring

In high-stakes cybersecurity environments like a Security Operations Center (SOC), automation is essential for both **access control enforcement** and **user provisioning**. This project simulates two common blue team challenges:

1. Detecting and responding to unauthorized login attempts based on IP address.
2. Generating unique employee IDs under strict security constraints.

Although built using basic Python structures, the logic behind this project is grounded in **SOC fundamentals** like access control lists (ACLs), authentication validation, and insider threat detection.

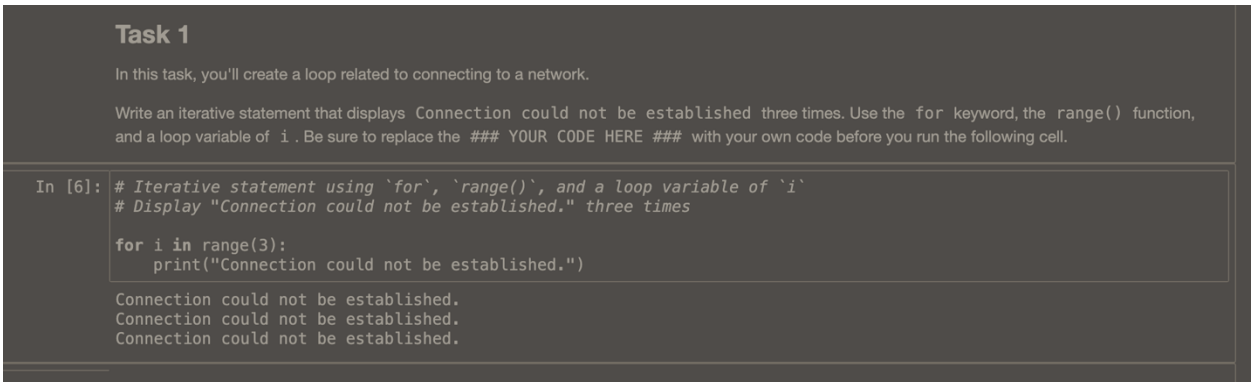
🔁 Phase 1: Network Access Verification and Alerting

🛠️ Task 1–3: Simulating Network Connection Attempts

To mimic failed connection attempts from unknown devices:

- I used a `for` loop and a `while` loop to print `"Connection could not be established"` three times.
- This represents repeated unauthorized access attempts — a key trigger in **SIEM alert logic**.

SOC Tie-in: This simulates real scenarios where brute-force or unauthorized access attempts are logged multiple times from the same host/IP. Detecting patterns like these is fundamental in tools like **Splunk**, **Sentinel**, or **Elastic SIEM**.



Task 1

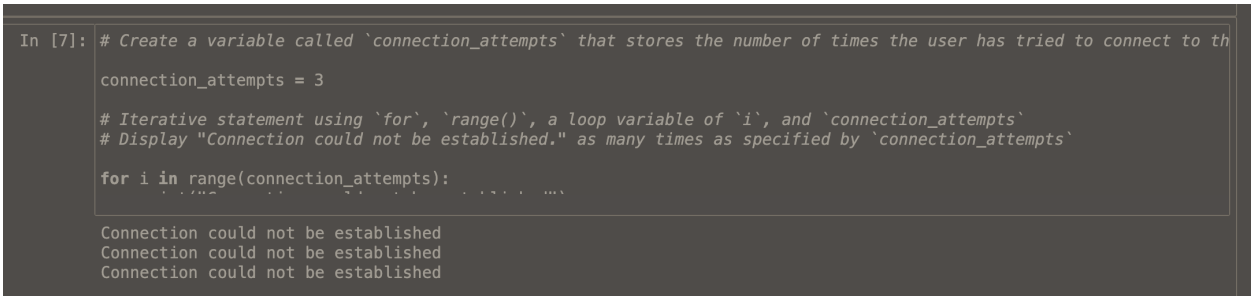
In this task, you'll create a loop related to connecting to a network.

Write an iterative statement that displays `Connection could not be established` three times. Use the `for` keyword, the `range()` function, and a loop variable of `i`. Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

```
In [6]: # Iterative statement using `for`, `range()`, and a loop variable of `i`
        # Display "Connection could not be established." three times

        for i in range(3):
            print("Connection could not be established.")
```

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



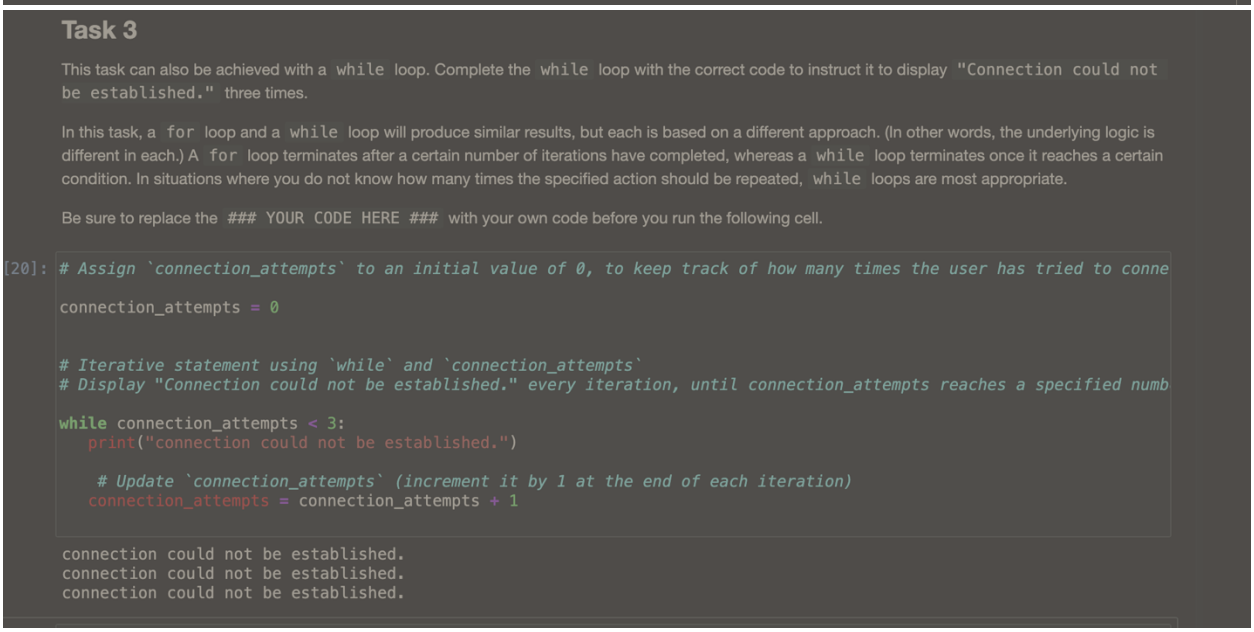
```
In [7]: # Create a variable called `connection_attempts` that stores the number of times the user has tried to connect to th

        connection_attempts = 3

        # Iterative statement using `for`, `range()`, a loop variable of `i`, and `connection_attempts`
        # Display "Connection could not be established." as many times as specified by `connection_attempts`

        for i in range(connection_attempts):
```

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



Task 3

This task can also be achieved with a `while` loop. Complete the `while` loop with the correct code to instruct it to display `"Connection could not be established."` three times.

In this task, a `for` loop and a `while` loop will produce similar results, but each is based on a different approach. (In other words, the underlying logic is different in each.) A `for` loop terminates after a certain number of iterations have completed, whereas a `while` loop terminates once it reaches a certain condition. In situations where you do not know how many times the specified action should be repeated, `while` loops are most appropriate.

Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

```
[20]: # Assign `connection_attempts` to an initial value of 0, to keep track of how many times the user has tried to conne

      connection_attempts = 0


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

      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.
```

🌐 Task 4–6: Validating IP Addresses Against an Allow List

🔎 *Setup:*

- Created a list of IPs attempting login (`ip_addresses`).
- Created an `allow_list` to act as a whitelist of permitted addresses.

✅ *What I Built:*

- A `for` loop that evaluates each IP.

- If an IP exists in the `allow_list`, it logs a clean message:
  `"IP address is allowed."`
- If not, it logs an escalation-worthy alert:
  `"IP address is not allowed. Further investigation of login activity required."`
- Used a `break` statement to **terminate the loop upon detecting suspicious access**, mimicking how many SOC playbooks **stop further processing and escalate** once a threat is identified.

Real-World Relevance: This logic mirrors **dynamic allow/deny list validation** in firewall rules, NAC (Network Access Control) solutions, and **SIEM enrichment pipelines** that auto-tag suspicious IPs.

Task 4

Now, you'll move onto your next task. You'll automate checking whether IP addresses are part of an allow list. You will start with a list of IP addresses from which users have tried to log in, stored in a variable called `ip_addresses`. Write a `for` loop that displays the elements of this list one at a time. Use `i` as the loop variable in the `for` loop.

Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

```
In [22]: # Assign `ip_addresses` to a list of IP addresses from which users have

         ip_addresses = ["192.168.142.245", "192.168.109.50", "192.168.86.232", "
                         "192.168.205.12", "192.168.200.48"]

         # For loop that displays the elements of `ip_addresses` one at a time

         for i in ip_addresses:
             print(i)
```

```
192.168.142.245
192.168.109.50
192.168.86.232
192.168.131.147
192.168.205.12
192.168.200.48
```

Task 5

You are now given a list of IP addresses that are allowed to log in, stored in a variable called `allow_list`. Write an `if` statement inside of the `for` loop. For each IP address in the list of IP addresses from which users have tried to log in, display `"IP address is allowed"` if it is among the allowed addresses and display `"IP address is not allowed"` otherwise.

Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

```
In [23]: # 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(i)
             else:
                 print("IP address is not allowed")
```

```
IP address is not allowed
IP address is not allowed
192.168.86.232
IP address is not allowed
192.168.205.12
IP address is not allowed
```

🧾 Phase 2: Secure and Compliant Employee ID Generation

👷 Task 7–8: Provisioning Unique Employee IDs

Goal:

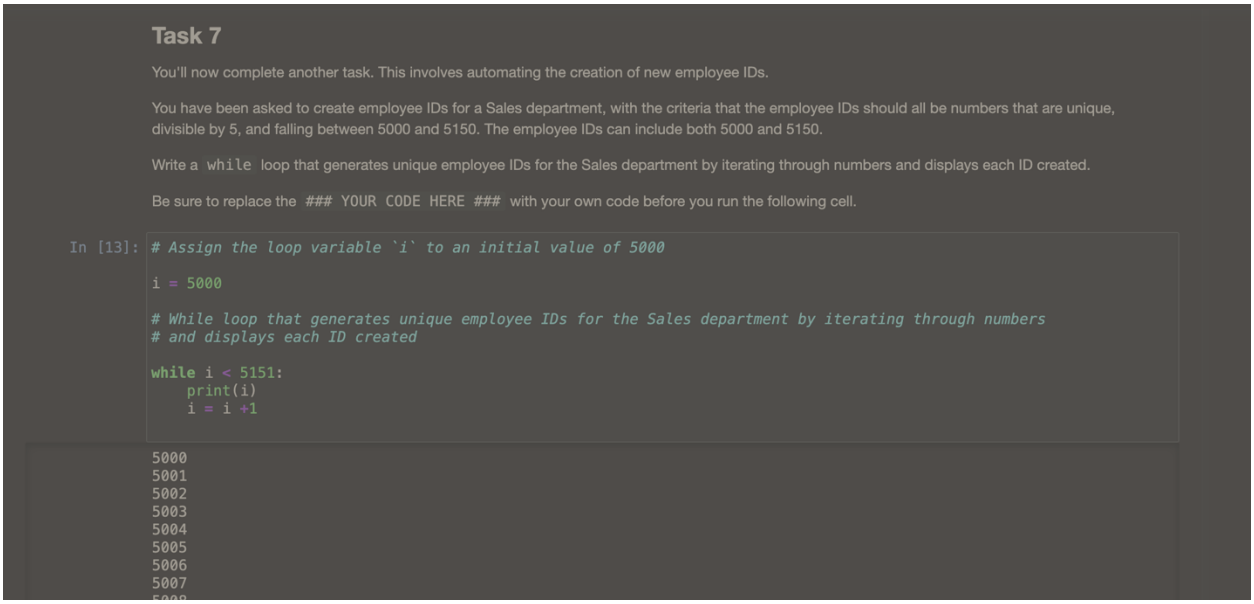
Generate employee IDs for the Sales department that meet the following security criteria:

- Unique
- Divisible by 5
- Fall within the range of **5000 to 5150** (inclusive)

Solution:

- Used a `while` loop to iterate over the ID range.
- Applied modulus (`%`) checks for divisibility.
- Added an `if` statement to trigger an alert when only 10 IDs remain, displaying: "⚠️ `Only 10 valid employee IDs remaining.`"

Transferable Skill: This demonstrates **secure provisioning logic**, useful in automating onboarding processes with ID rules that reduce collision risk and maintain compliance.



Task 7

You'll now complete another task. This involves automating the creation of new employee IDs.

You have been asked to create employee IDs for a Sales department, with the criteria that the employee IDs should all be numbers that are unique, divisible by 5, and falling between 5000 and 5150. The employee IDs can include both 5000 and 5150.

Write a `while` loop that generates unique employee IDs for the Sales department by iterating through numbers and displays each ID created.

Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

In [13]:
```
# 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

while i < 5151:
    print(i)
    i = i +1
```

```
5000
5001
5002
5003
5004
5005
5006
5007
```

Task 6

Imagine now that the information the users are trying to access is restricted, and if an IP address outside the list of allowed IP addresses attempts access, the loop should terminate because further investigation would be needed to assess whether this activity poses a threat. To achieve this, use the `break` keyword and expand the message that is displayed to the user when their IP address is not in `allow_list` to provide more specifics. Instead of `"IP address is not allowed"`, display `"IP address is not allowed. Further investigation of login activity required"`.

Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

n [12]:
```
# 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"]

# Check each login attempt
for i in ip_addresses:
    if i in allow_list:
        print(f"{i} is allowed")
    else:
        print(f"{i} is not allowed. Further investigation of login activity required")
        break
```

```
192.168.142.245 is not allowed. Further investigation of login activity required
```

Task 8

You would like to incorporate a message that displays `Only 10 valid employee ids remaining` as a helpful alert once the loop variable reaches `5100`.

To do so, include an `if` statement in your code.

Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

In [1]:
```
i = 5000

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

```
5000
5005
5010
5015
5020
5025
5030
5035
5040
5045
5050
5055
5060
5065
5070
5075
5080
5085
5090
5095
5100
Only 10 valid employee IDs remaining
5105
5110
5115
5120
5125
5130
5135
5140
5145
5150
```

```
# While loop that generates unique employee IDs for the Sales department by iterating through numbers
# and displays each ID created

while i < 5151:
    print(i)
    i = i + 1
```

```
5132
5133
5134
5135
5136
5137
5138
5139
5140
5141
5142
5143
5144
5145
5146
5147
5148
5149
5150
```

🎯 Results & Reflections

This project wasn't just a Python exercise — it was a **controlled simulation of critical SOC workflows**, including:

Security Objective	Implementation Example
Unauthorized Access Detection	Loop-based IP validation with alert escalation (`break`)
Access Control Enforcement	`if` conditionals to mimic dynamic ACLs
Secure Identity Provisioning	Loop logic for controlled employee ID generation
Proactive Alerting & Logging	Conditional alert messaging when nearing ID exhaustion

---

🧰 Tools & Skills Demonstrated

- **Python (Core Scripting)**: Loops, conditionals, modular logic
- **Security Automation Thinking**: Simulated SOC workflows and escalation points
- **Access Control Logic**: Allow list validation with early termination
- **Alerting Strategy**: Proactive user feedback tied to loop thresholds

---