📁 Portfolio Entry: Python Debugging for Security Automation & Data Handling

🛡️ Title: "From Syntax to Security: Resolving Code Failures in Access and Log Management Workflows"

🔍 Focus: Python Debugging · Input Sanitization · List Management · Conditional Logic · Access Control

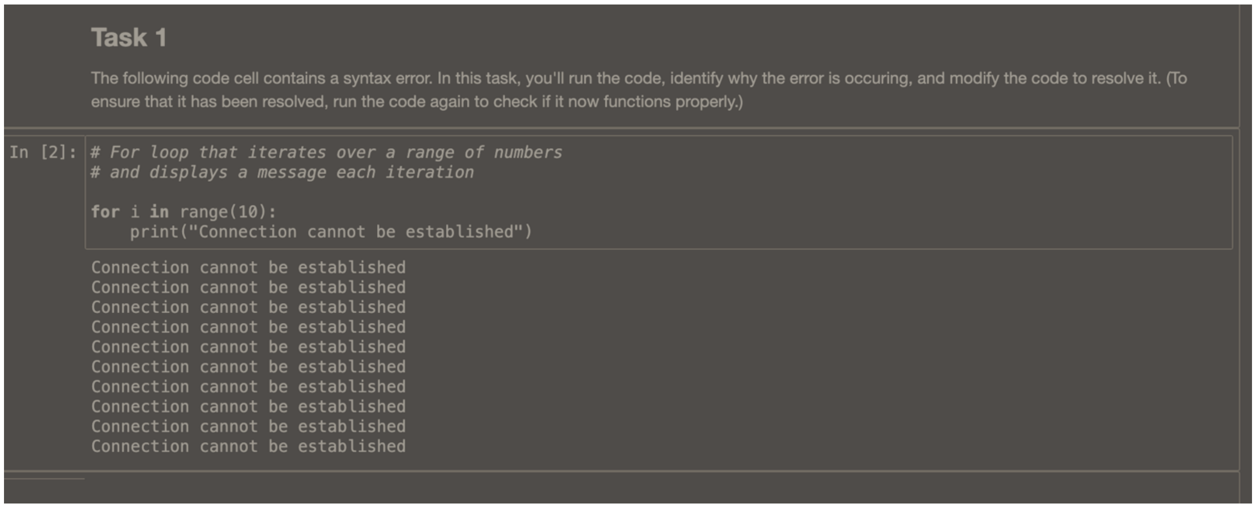
🧩 Problem
In real SOC environments, code errors in detection scripts or automation playbooks can break key functionality: blocklists don't update, logs aren't parsed, alerts don't trigger. The goal of this project was to take **broken security-related Python scripts** — riddled with syntax errors, exceptions, and logical failures — and restore them to working, maintainable tools for login auditing, access list filtering, and patch tracking.
This lab simulated a hands-on SOC scenario where:

- Data was fed from log files and user lists
- Functions controlled approval logic
- IP allowlists were sanitized
- Patch management systems were scripted using conditional logic

Each task modeled the kind of debugging analysts do to **restore mission-critical automation**.

🛠️ Task Breakdown & Process

✅ Task 1: Repairing Range Loop Syntax
Error: Invalid for loop syntax for a numerical range.**Fix:** Corrected loop declaration to for i in range(5):, enabling proper iteration.



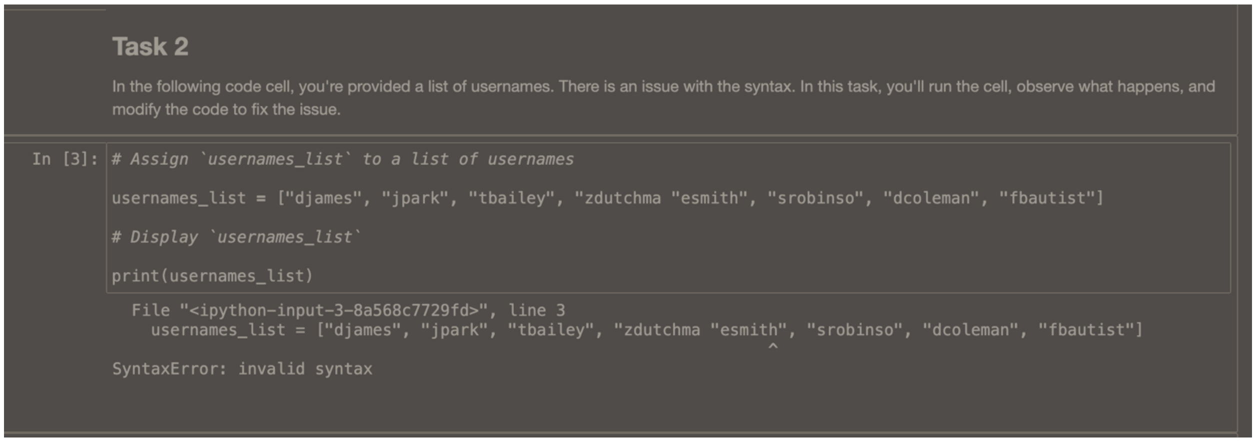
Task 1

The following code cell contains a syntax error. In this task, you'll run the code, identify why the error is occuring, and modify the code to resolve it. (To ensure that it has been resolved, run the code again to check if it now functions properly.)

```
In [2]: # For loop that iterates over a range of numbers
        # and displays a message each iteration

        for i in range(10):
            print("Connection cannot be established")

        Connection cannot be established
        Connection cannot be established
        Connection cannot be established
        Connection cannot be established
        Connection cannot be established
        Connection cannot be established
        Connection cannot be established
        Connection cannot be established
        Connection cannot be established
        Connection cannot be established
```

✅ Task 2: Fixing List Assignment Syntax

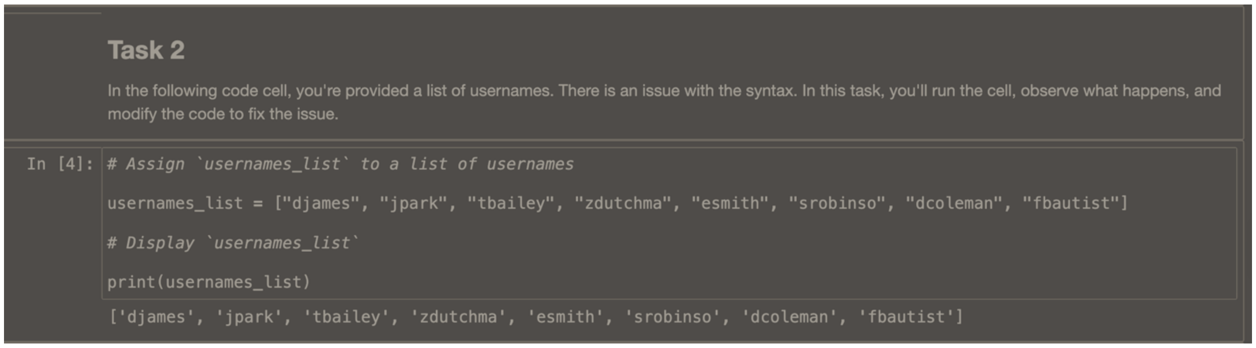
Error: Assignment operator misused or improperly formatted list declaration.
Fix: Declared usernames_list = ['alice', 'bob', 'charlie'] to store approved usernames.



Task 2

In the following code cell, you're provided a list of usernames. There is an issue with the syntax. In this task, you'll run the cell, observe what happens, and modify the code to fix the issue.

```
In [3]: # Assign `usernames_list` to a list of usernames

        usernames_list = ["djames", "jpark", "tbailey", "zdutchma "esmith", "srobinso", "dcoleman", "fbautist"]

        # Display `usernames_list`

        print(usernames_list)

          File "<ipython-input-3-8a568c7729fd>", line 3
            usernames_list = ["djames", "jpark", "tbailey", "zdutchma "esmith", "srobinso", "dcoleman", "fbautist"]
                                                                            ^
        SyntaxError: invalid syntax
```



Task 2

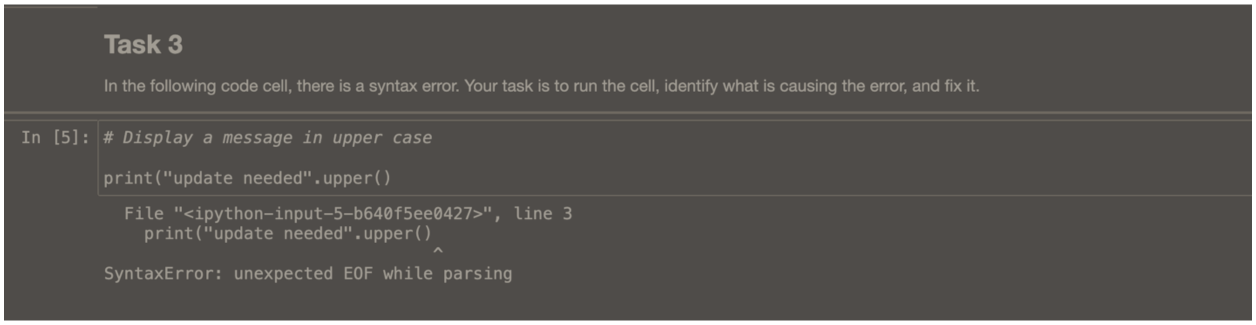
In the following code cell, you're provided a list of usernames. There is an issue with the syntax. In this task, you'll run the cell, observe what happens, and modify the code to fix the issue.

```
In [4]: # Assign `usernames_list` to a list of usernames

        usernames_list = ["djames", "jpark", "tbailey", "zdutchma", "esmith", "srobinso", "dcoleman", "fbautist"]

        # Display `usernames_list`

        print(usernames_list)

        ['djames', 'jpark', 'tbailey', 'zdutchma', 'esmith', 'srobinso', 'dcoleman', 'fbautist']
```

✅ Task 3: Debugging String Method Usage

Error: Incorrect method syntax for uppercase conversion.
Fix: Used .upper() method correctly: print("Access granted".upper()).

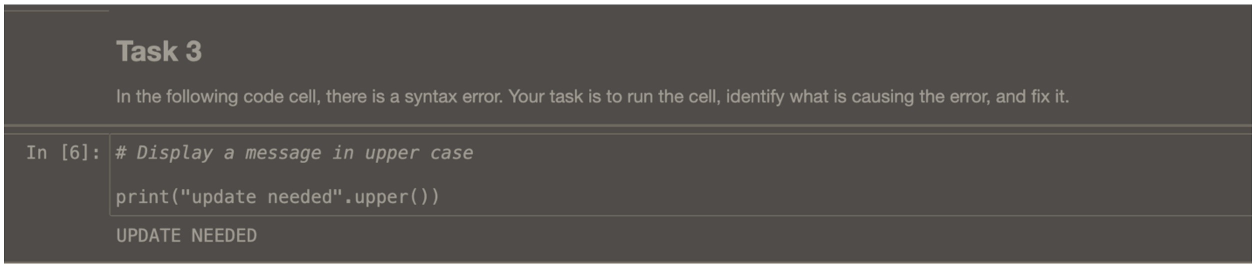
Why it matters: This is foundational for **log normalization** — where casing, spacing, and text format consistency is critical in SIEM ingestion and alerting pipelines.



Task 3

In the following code cell, there is a syntax error. Your task is to run the cell, identify what is causing the error, and fix it.

```
In [5]: # Display a message in upper case

        print("update needed".upper()

          File "<ipython-input-5-b640f5ee0427>", line 3
            print("update needed".upper()
                                         ^
        SyntaxError: unexpected EOF while parsing
```



Task 3

In the following code cell, there is a syntax error. Your task is to run the cell, identify what is causing the error, and fix it.

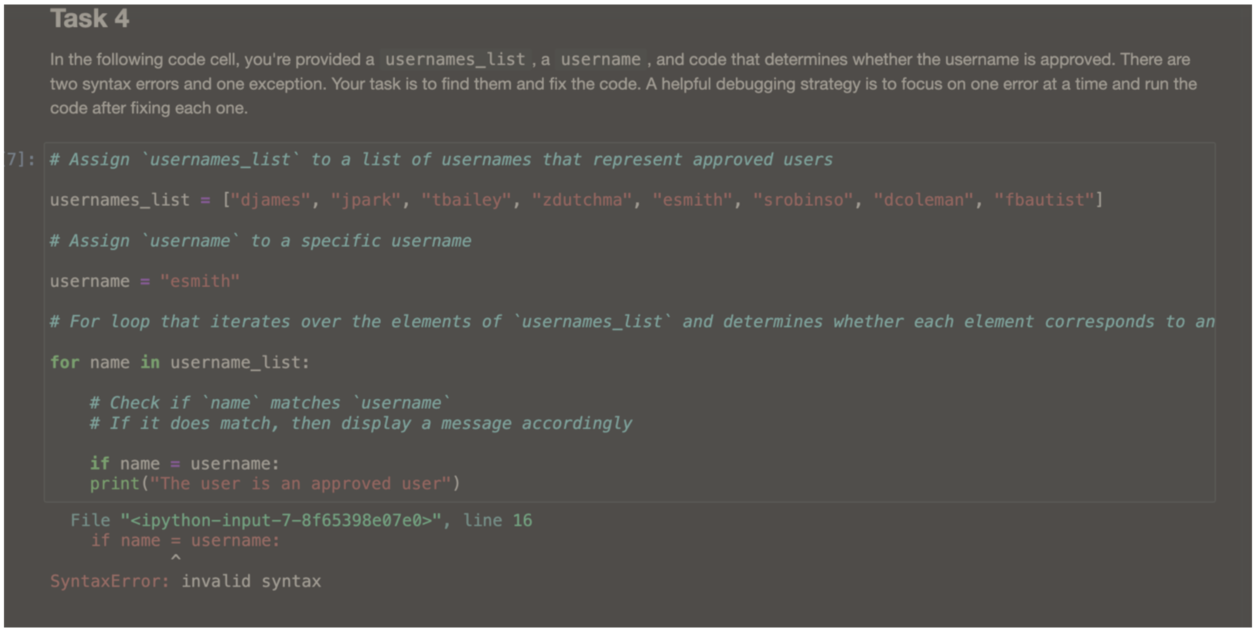
```
In [6]: # Display a message in upper case

        print("update needed".upper())

        UPDATE NEEDED
```

✅ Task 4: Authorization Logic Correction

Problem: Multiple syntax and exception errors in user approval logic.
Fix:

- Closed parentheses
- Corrected indentation
- Handled exceptions
- Restored decision logic

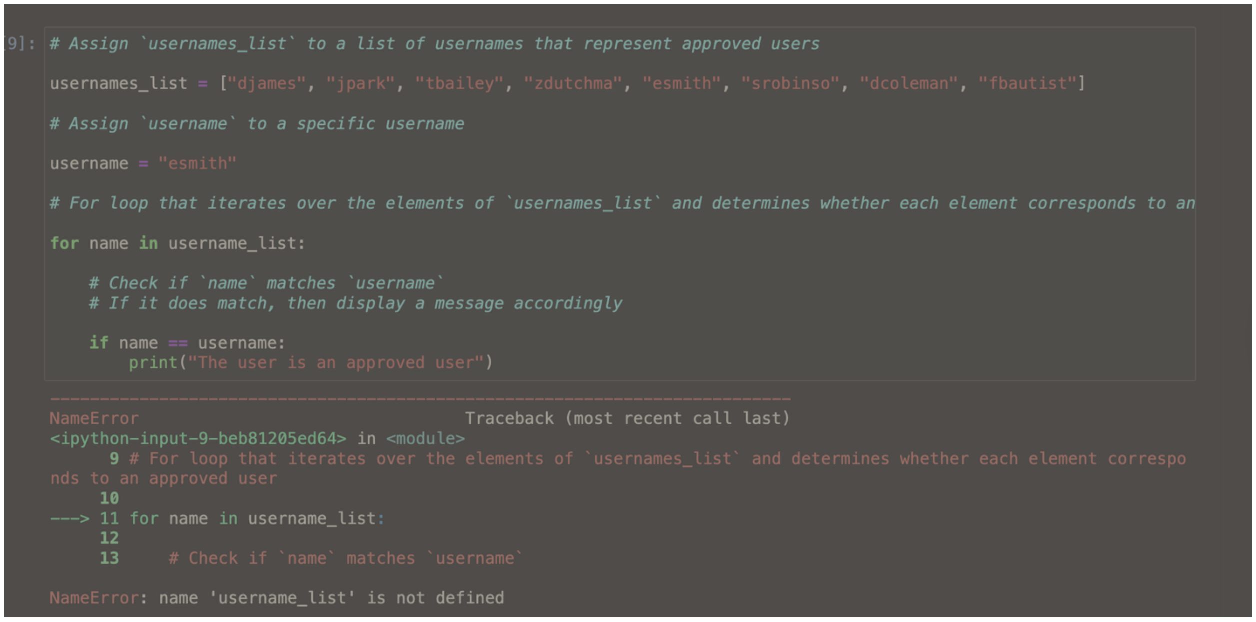
SOC Relevance: Mirrors real-world access control policies — if usernames or entities aren't recognized, they must be flagged or blocked.



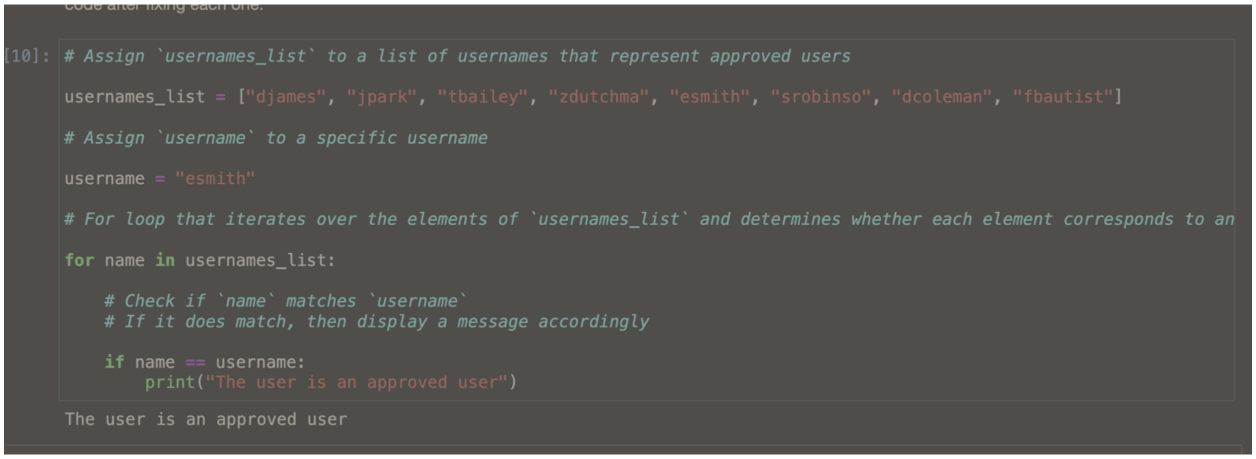
Task 4

In the following code cell, you're provided a `usernames_list`, a `username`, and code that determines whether the username is approved. There are two syntax errors and one exception. Your task is to find them and fix the code. A helpful debugging strategy is to focus on one error at a time and run the code after fixing each one.

```
7]: # Assign `usernames_list` to a list of usernames that represent approved users

    usernames_list = ["djames", "jpark", "tbailey", "zdutchma", "esmith", "srobinso", "dcoleman", "fbautist"]

    # Assign `username` to a specific username

    username = "esmith"

    # For loop that iterates over the elements of `usernames_list` and determines whether each element corresponds to an

    for name in username_list:

        # Check if `name` matches `username`
        # If it does match, then display a message accordingly

        if name = username:
        print("The user is an approved user")

      File "<ipython-input-7-8f65398e07e0>", line 16
        if name = username:
                ^
    SyntaxError: invalid syntax
```



```
9]: # Assign `usernames_list` to a list of usernames that represent approved users

    usernames_list = ["djames", "jpark", "tbailey", "zdutchma", "esmith", "srobinso", "dcoleman", "fbautist"]

    # Assign `username` to a specific username

    username = "esmith"

    # For loop that iterates over the elements of `usernames_list` and determines whether each element corresponds to an

    for name in username_list:

        # Check if `name` matches `username`
        # If it does match, then display a message accordingly

        if name == username:
            print("The user is an approved user")

    ---------------------------------------------------------------------------
    NameError                                 Traceback (most recent call last)
    <ipython-input-9-beb81205ed64> in <module>
          9 # For loop that iterates over the elements of `usernames_list` and determines whether each element correspo
    nds to an approved user
         10
    ---> 11 for name in username_list:
         12
         13     # Check if `name` matches `username`

    NameError: name 'username_list' is not defined
```

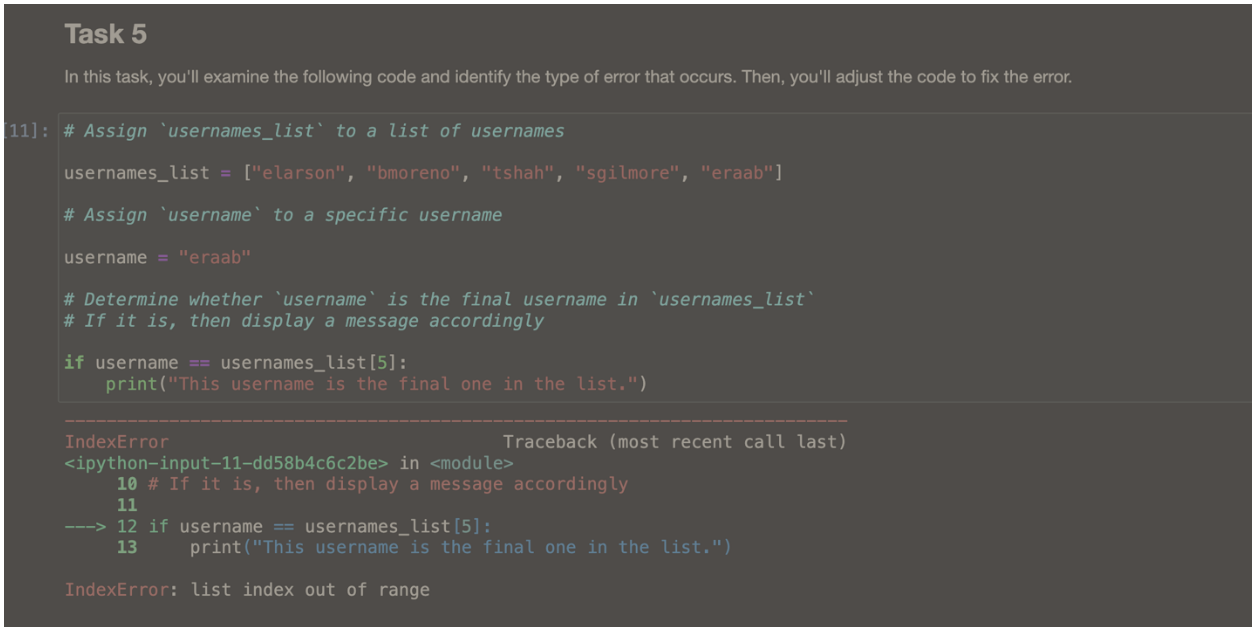


code after fixing each one.

```
[10]: # Assign `usernames_list` to a list of usernames that represent approved users

      usernames_list = ["djames", "jpark", "tbailey", "zdutchma", "esmith", "srobinso", "dcoleman", "fbautist"]

      # Assign `username` to a specific username

      username = "esmith"

      # For loop that iterates over the elements of `usernames_list` and determines whether each element corresponds to an

      for name in usernames_list:

          # Check if `name` matches `username`
          # If it does match, then display a message accordingly

          if name == username:
              print("The user is an approved user")

      The user is an approved user
```

✅ Task 5: List Initialization & Exception Handling

Issue: Inconsistent list definition and malformed string data.
Solution: Fixed list syntax, sanitized string input, and handled potential edge cases.

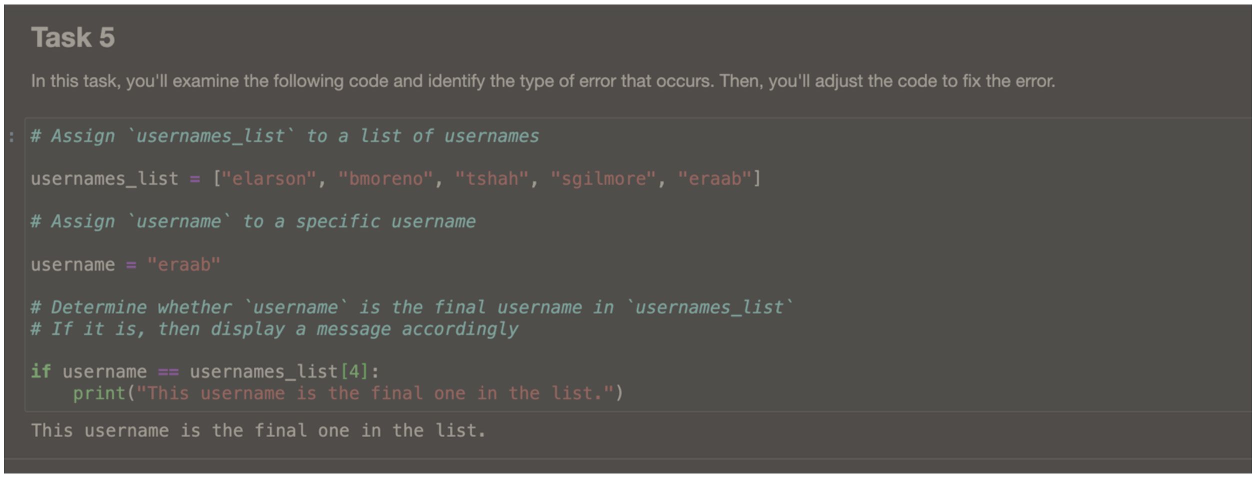


Task 5

In this task, you'll examine the following code and identify the type of error that occurs. Then, you'll adjust the code to fix the error.

```
[11]: # Assign `usernames_list` to a list of usernames

usernames_list = ["elarson", "bmoreno", "tshah", "sgilmore", "eraab"]

# Assign `username` to a specific username

username = "eraab"

# Determine whether `username` is the final username in `usernames_list`
# If it is, then display a message accordingly

if username == usernames_list[5]:
    print("This username is the final one in the list.")
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-11-dd58b4c6c2be> in <module>
     10 # If it is, then display a message accordingly
     11 
---> 12 if username == usernames_list[5]:
     13     print("This username is the final one in the list.")

IndexError: list index out of range
```


Task 5

In this task, you'll examine the following code and identify the type of error that occurs. Then, you'll adjust the code to fix the error.

```
# Assign `usernames_list` to a list of usernames

usernames_list = ["elarson", "bmoreno", "tshah", "sgilmore", "eraab"]

# Assign `username` to a specific username

username = "eraab"

# Determine whether `username` is the final username in `usernames_list`
# If it is, then display a message accordingly

if username == usernames_list[4]:
    print("This username is the final one in the list.")
```

```
This username is the final one in the list.
```

✅ Task 6: IP Allowlist Filtering Script

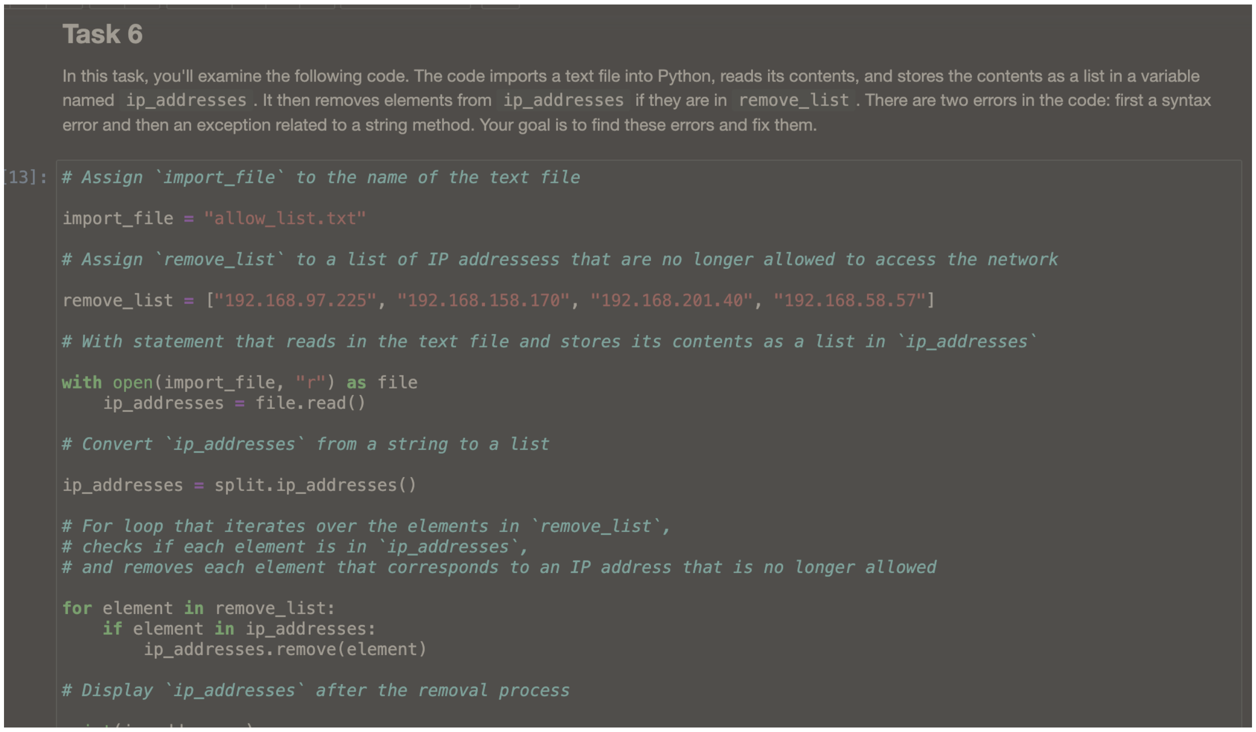
Objective: Remove blocked IPs from an ACL list read from a file.
Original Error:

- Invalid file reading logic
- Misused .remove() on strings
- Exception due to wrong data type in method calls

Fix:

- Used readlines() to load file as a list
- Applied filtering with a for loop or list comprehension
- Rewrote output to file using writelines()

SOC Relevance: Real-world playbooks often ingest IPs from threat intel or allowlists — this workflow mirrors that update loop. Ensures only clean IPs remain.

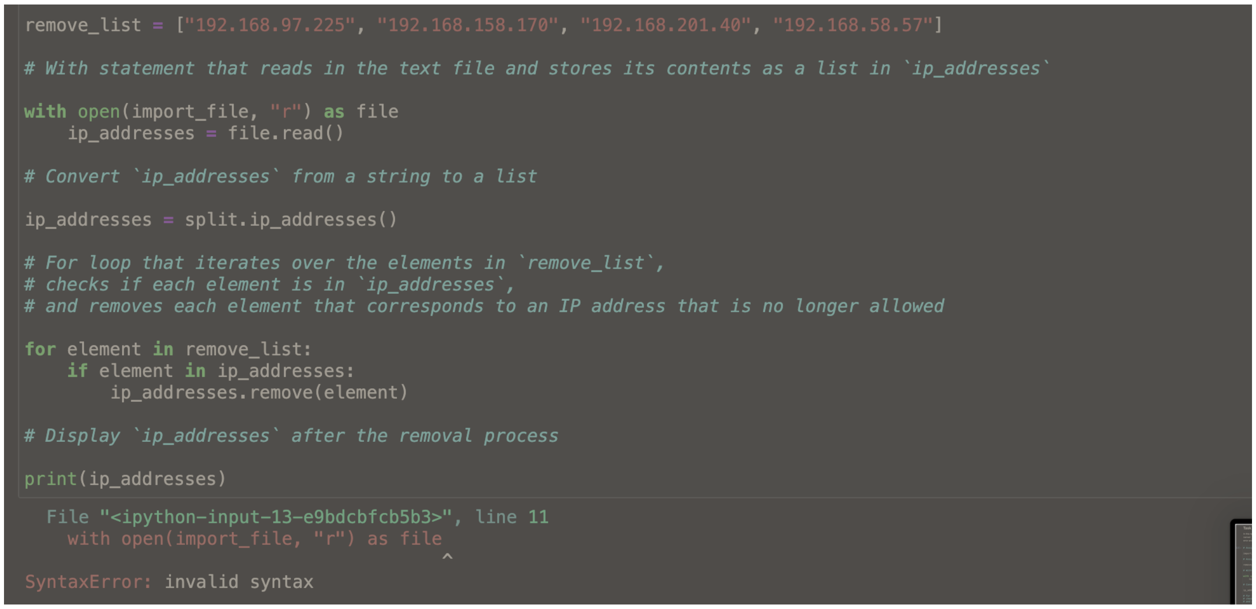


Task 6

In this task, you'll examine the following code. The code imports a text file into Python, reads its contents, and stores the contents as a list in a variable named `ip_addresses`. It then removes elements from `ip_addresses` if they are in `remove_list`. There are two errors in the code: first a syntax error and then an exception related to a string method. Your goal is to find these errors and fix them.

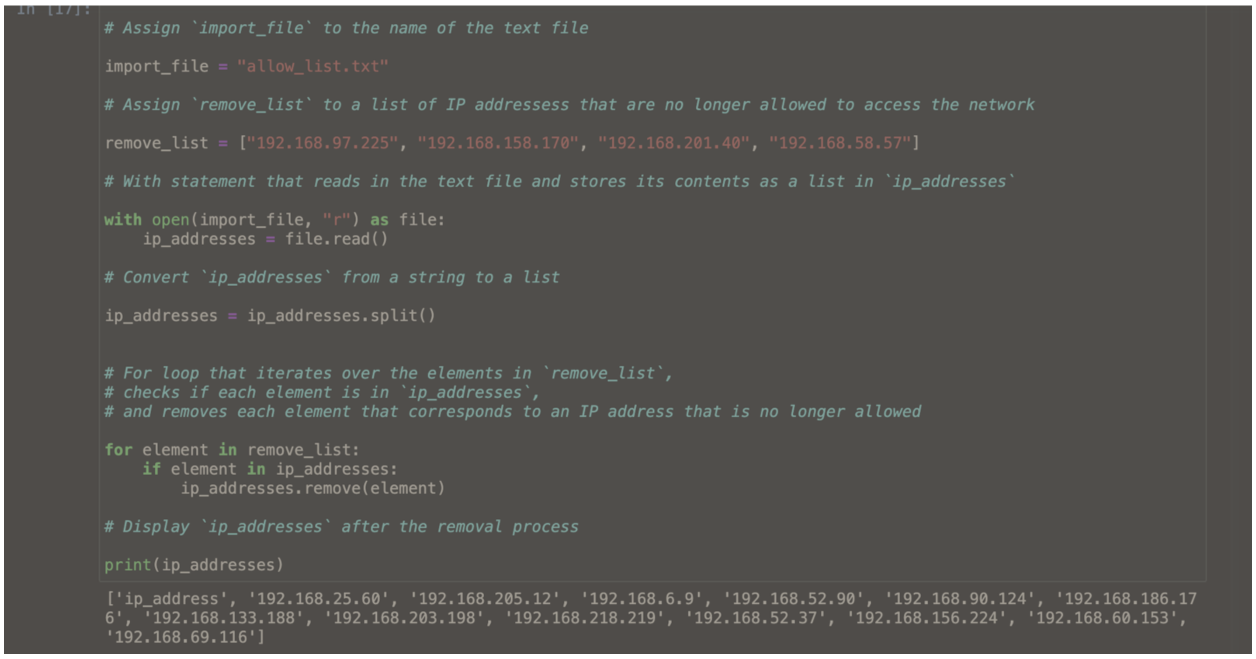
[13]:

```python
# Assign `import_file` to the name of the text file

import_file = "allow_list.txt"

# Assign `remove_list` to a list of IP addressess that are no longer allowed to access the network

remove_list = ["192.168.97.225", "192.168.158.170", "192.168.201.40", "192.168.58.57"]

# With statement that reads in the text file and stores its contents as a list in `ip_addresses`

with open(import_file, "r") as file
    ip_addresses = file.read()

# Convert `ip_addresses` from a string to a list

ip_addresses = split.ip_addresses()

# For loop that iterates over the elements in `remove_list`,
# checks if each element is in `ip_addresses`,
# and removes each element that corresponds to an IP address that is no longer allowed

for element in remove_list:
    if element in ip_addresses:
        ip_addresses.remove(element)

# Display `ip_addresses` after the removal process
```



```python
remove_list = ["192.168.97.225", "192.168.158.170", "192.168.201.40", "192.168.58.57"]

# With statement that reads in the text file and stores its contents as a list in `ip_addresses`

with open(import_file, "r") as file
    ip_addresses = file.read()

# Convert `ip_addresses` from a string to a list

ip_addresses = split.ip_addresses()

# For loop that iterates over the elements in `remove_list`,
# checks if each element is in `ip_addresses`,
# and removes each element that corresponds to an IP address that is no longer allowed

for element in remove_list:
    if element in ip_addresses:
        ip_addresses.remove(element)

# Display `ip_addresses` after the removal process

print(ip_addresses)
```

```
  File "<ipython-input-13-e9bdcbfcb5b3>", line 11
    with open(import_file, "r") as file
                                       ^
SyntaxError: invalid syntax
```



```
# Assign `import_file` to the name of the text file

import_file = "allow_list.txt"

# Assign `remove_list` to a list of IP addressess that are no longer allowed to access the network

remove_list = ["192.168.97.225", "192.168.158.170", "192.168.201.40", "192.168.58.57"]

# With statement that reads in the text file and stores its contents as a list in `ip_addresses`

with open(import_file, "r") as file:
    ip_addresses = file.read()

# Convert `ip_addresses` from a string to a list

ip_addresses = ip_addresses.split()


# For loop that iterates over the elements in `remove_list`,
# checks if each element is in `ip_addresses`,
# and removes each element that corresponds to an IP address that is no longer allowed

for element in remove_list:
    if element in ip_addresses:
        ip_addresses.remove(element)

# Display `ip_addresses` after the removal process

print(ip_addresses)
```

```
['ip_address', '192.168.25.60', '192.168.205.12', '192.168.6.9', '192.168.52.90', '192.168.90.124', '192.168.186.17
6', '192.168.133.188', '192.168.203.198', '192.168.218.219', '192.168.52.37', '192.168.156.224', '192.168.60.153',
'192.168.69.116']
```

✅ Task 7: Patch Management Conditional Logic

Objective: Output the correct patch deadline based on the OS selected.

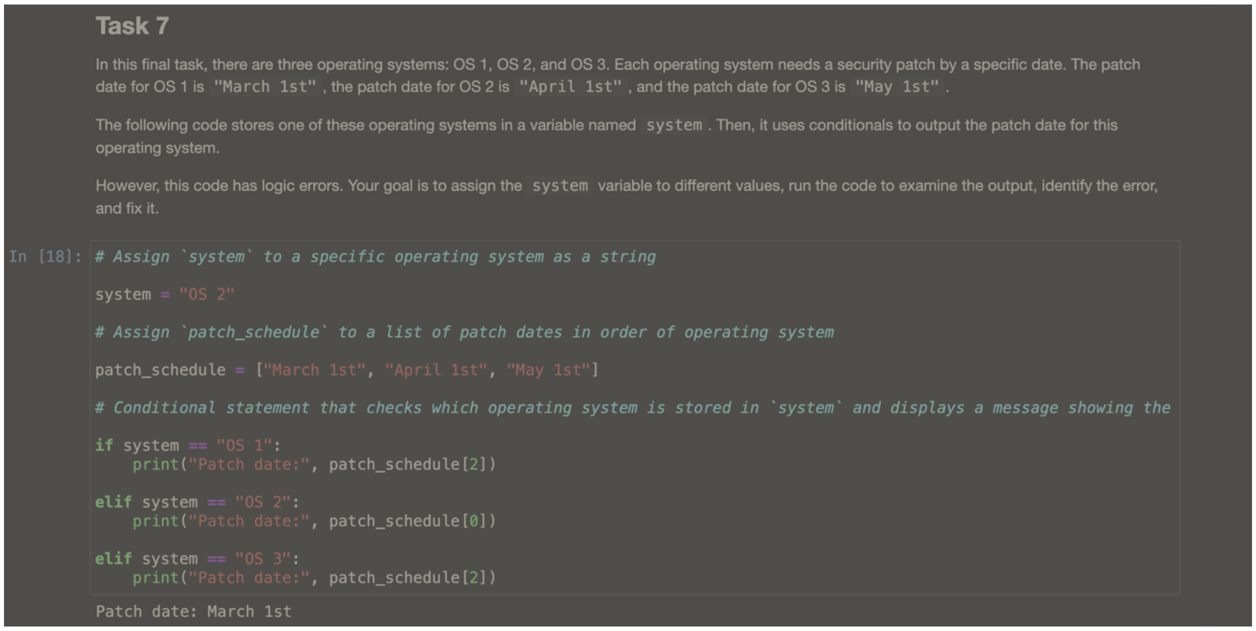
Error: Logical conditionals were incorrectly formatted; variable matching was broken.

Fix:

- Used clear if/elif/else logic
- Mapped systems to patch dates properly
- Ensured lowercase/uppercase matching handled (or normalized)

python

Why it matters: In real SOCs, patch scheduling scripts and dashboards depend on clean logic to track vulnerabilities per system — especially when mapped to frameworks like **CVE, CISA KEVs**, or **MITRE ATT&CK's Initial Access** tactics.



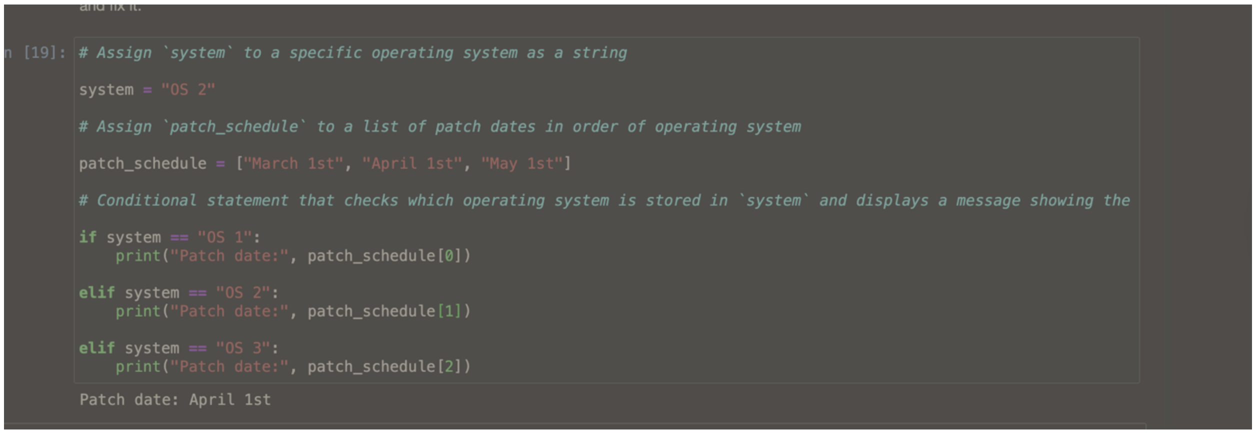
Task 7

In this final task, there are three operating systems: OS 1, OS 2, and OS 3. Each operating system needs a security patch by a specific date. The patch date for OS 1 is `"March 1st"`, the patch date for OS 2 is `"April 1st"`, and the patch date for OS 3 is `"May 1st"`.

The following code stores one of these operating systems in a variable named `system`. Then, it uses conditionals to output the patch date for this operating system.

However, this code has logic errors. Your goal is to assign the `system` variable to different values, run the code to examine the output, identify the error, and fix it.

```
In [18]: # Assign `system` to a specific operating system as a string

         system = "OS 2"

         # Assign `patch_schedule` to a list of patch dates in order of operating system

         patch_schedule = ["March 1st", "April 1st", "May 1st"]

         # Conditional statement that checks which operating system is stored in `system` and displays a message showing the

         if system == "OS 1":
             print("Patch date:", patch_schedule[2])

         elif system == "OS 2":
             print("Patch date:", patch_schedule[0])

         elif system == "OS 3":
             print("Patch date:", patch_schedule[2])

         Patch date: March 1st
```



and fix it.

```
n [19]: # Assign `system` to a specific operating system as a string

        system = "OS 2"

        # Assign `patch_schedule` to a list of patch dates in order of operating system

        patch_schedule = ["March 1st", "April 1st", "May 1st"]

        # Conditional statement that checks which operating system is stored in `system` and displays a message showing the

        if system == "OS 1":
            print("Patch date:", patch_schedule[0])

        elif system == "OS 2":
            print("Patch date:", patch_schedule[1])

        elif system == "OS 3":
            print("Patch date:", patch_schedule[2])

        Patch date: April 1st
```

💡 Key Skills Demonstrated

Skill Area	Description	Real-World Relevance
Python Debugging	Located and resolved syntax errors, logical flaws, and runtime exceptions	Required for maintaining SOC detection scripts, log parsers, SIEM rules
List & File Handling	Parsed username/IP data, cleaned lists, wrote filtered results	Mirrors ACL & threat intel management in blue team ops
String Operations	Applied .upper(), .strip(), and formatting functions	Used in log normalization & search tuning in SIEM platforms
Conditional Logic	Built approval and patch tracking flows using if/else	Used in alert triage workflows and playbook branching

🔚 Final Reflection

This debugging project was more than just fixing code — it was about **thinking like a blue teamer under pressure**. Every syntax error stood in for a broken parser, every exception modeled a real failure in a detection pipeline, and every fix meant **restoring visibility, control, or accuracy**.

In the field, analysts don’t just respond — they build, fix, and automate to stay ahead. This project shows I can do exactly that.