Algorithm for file updates in Python

Project description

I am acting as a security professional at a healthcare company responsible for managing access to restricted content based on employee IP addresses. Employees are allowed access through an "allow list," but this list must be regularly updated to reflect changes, such as removing IP addresses identified on a "remove list." The task involves creating an algorithm to check if any IP addresses in the allow list also appear on the remove list and then remove them from the allow list.

Open the file that contains the allowed list

To open the file containing the allow list, the code uses a `with` statement and the `open()` function in Python.

Python Syntax and Functions:

- **`with open()`**: This is a context manager used to open a file safely. It ensures the file is properly closed after operations are completed, even if an error occurs.
- **`"r"` mode**: The `"r"` mode opens the file in read-only mode, allowing the program to access its contents without modifying them.
- **`import_file`**: The variable `import_file` stores the name of the file (`allow_list.txt`), which contains the list of approved IP addresses.

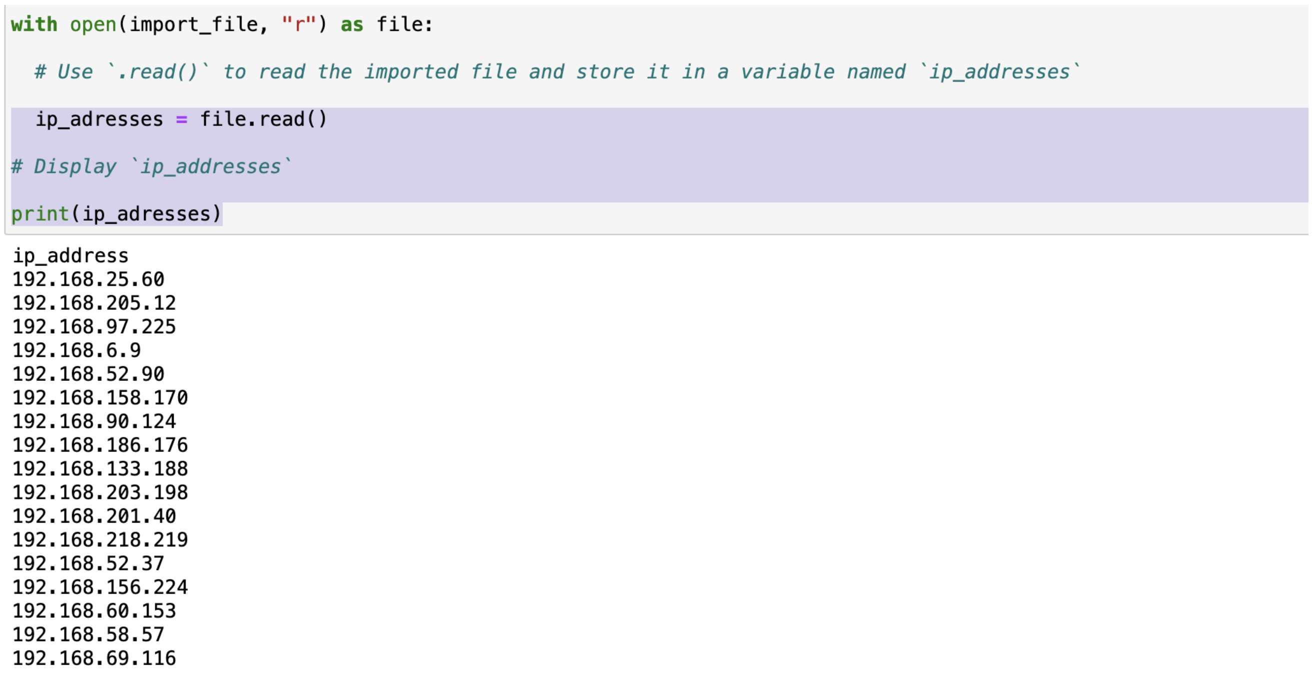
```
# Assign `import_file` to the name of the file

import_file = "allow_list.txt"

# Assign `remove_list` to a list of IP addresses that are no longer allowed to access restricted information.

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

# First line of `with` statement

with open(import_file, "r") as file:
```

Read the file contents

In this step, the program reads the contents of the `allow_list.txt` file and stores them in a variable named `ip_addresses`.

Python Syntax and Functions:

- `with open(import_file, "r")`: Opens the file in read-only mode using a context manager, ensuring safe file handling.
- `.read()`: This method reads the entire content of the file as a single string. The contents are then stored in the variable `ip_addresses`.
- `print()`: Displays the contents of the `ip_addresses` variable, allowing me to verify what has been read from the file.



```
with open(import_file, "r") as file:

  # Use `.read()` to read the imported file and store it in a variable named `ip_addresses`

  ip_adresses = file.read()

# Display `ip_addresses`

print(ip_adresses)
```

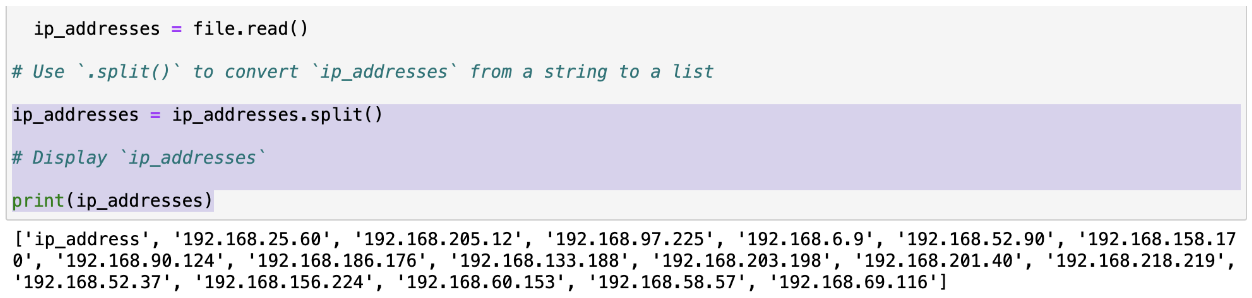
```
ip_address
192.168.25.60
192.168.205.12
192.168.97.225
192.168.6.9
192.168.52.90
192.168.158.170
192.168.90.124
192.168.186.176
192.168.133.188
192.168.203.198
192.168.201.40
192.168.218.219
192.168.52.37
192.168.156.224
192.168.60.153
192.168.58.57
192.168.69.116
```

Convert the string into a list

This step converts the contents of the `ip_addresses` variable from a single string into a list of individual IP addresses.

Python Syntax and Functions:

- `.split()`: The `.split()` method divides a string into a list of substrings based on a delimiter. By default, it uses whitespace (spaces, tabs, newlines) as the delimiter.
  - In this case, the `ip_addresses` string is split into a list of individual IP addresses.
- **Reassignment to `ip_addresses`**: The variable is reassigned with the result of `.split()` to store the IP addresses as a list.



```
  ip_addresses = file.read()

# Use `.split()` to convert `ip_addresses` from a string to a list

ip_addresses = ip_addresses.split()

# Display `ip_addresses`

print(ip_addresses)
```

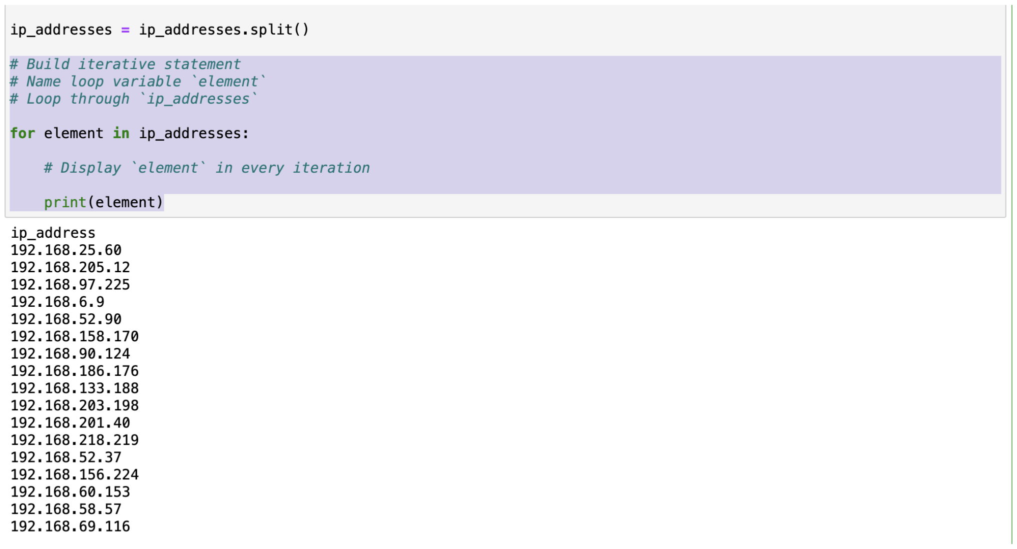
```
['ip_address', '192.168.25.60', '192.168.205.12', '192.168.97.225', '192.168.6.9', '192.168.52.90', '192.168.158.17
0', '192.168.90.124', '192.168.186.176', '192.168.133.188', '192.168.203.198', '192.168.201.40', '192.168.218.219',
'192.168.52.37', '192.168.156.224', '192.168.60.153', '192.168.58.57', '192.168.69.116']
```

Iterate through the remove list

This step iterates through the `ip_addresses` list, allowing me to process or inspect each individual IP address.

Python Syntax and Functions:

- **`for element in ip_addresses:`**
  - This is a `for` loop that iterates over each item (IP address) in the `ip_addresses` list.
  - Each item is temporarily stored in the variable `element` during each iteration.
- **`print(element)`**
  - This outputs the current value of `element` (the current IP address) during each iteration, allowing me to view each address individually.



```
ip_addresses = ip_addresses.split()

# Build iterative statement
# Name loop variable `element`
# Loop through `ip_addresses`

for element in ip_addresses:

    # Display `element` in every iteration

    print(element)
```

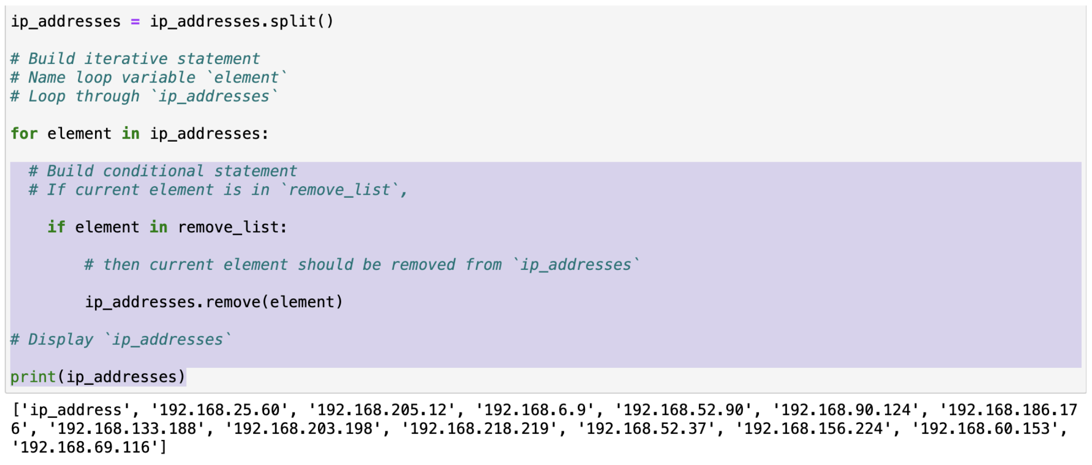
```
ip_address
192.168.25.60
192.168.205.12
192.168.97.225
192.168.6.9
192.168.52.90
192.168.158.170
192.168.90.124
192.168.186.176
192.168.133.188
192.168.203.198
192.168.201.40
192.168.218.219
192.168.52.37
192.168.156.224
192.168.60.153
192.168.58.57
192.168.69.116
```

Remove IP addresses that are on the remove list

This step identifies and removes any IP addresses in the `ip_addresses` list also present in the `remove_list`.

Python Syntax and Functions:

1. **`if element in remove_list:`**
   - This conditional statement checks if the current IP address (`element`) is present in the `remove_list`.
   - If the condition is `True`, the IP address needs to be removed from the `ip_addresses` list.
2. **`ip_addresses.remove(element)`**
   - The `.remove()` method removes the first occurrence of the specified value (`element`) from the list.
   - This ensures that restricted IP addresses are dynamically removed from the allow list.
3. **`print(ip_addresses)`**
   - After the loop completes, this displays the updated `ip_addresses` list, now free of restricted IPs.



```
ip_addresses = ip_addresses.split()

# Build iterative statement
# Name loop variable `element`
# Loop through `ip_addresses`

for element in ip_addresses:

  # Build conditional statement
  # If current element is in `remove_list`,

    if element in remove_list:

        # then current element should be removed from `ip_addresses`

        ip_addresses.remove(element)

# Display `ip_addresses`

print(ip_addresses)
```

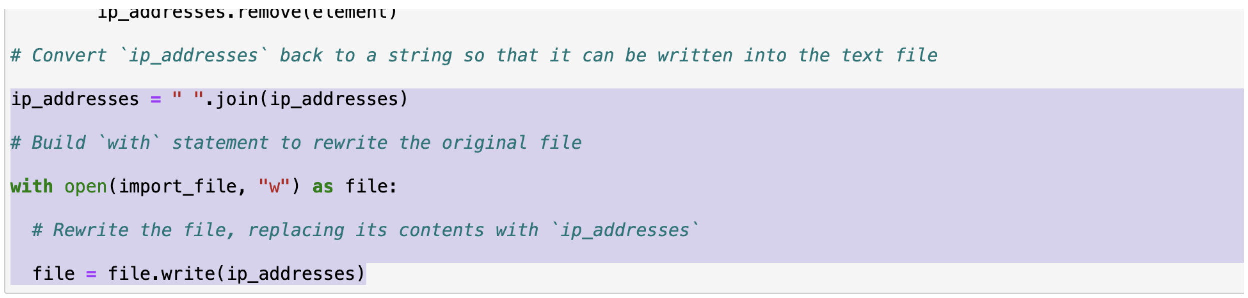
```
['ip_address', '192.168.25.60', '192.168.205.12', '192.168.6.9', '192.168.52.90', '192.168.90.124', '192.168.186.17
6', '192.168.133.188', '192.168.203.198', '192.168.218.219', '192.168.52.37', '192.168.156.224', '192.168.60.153',
'192.168.69.116']
```

Update the file with the revised list of IP addresses

This step writes the updated `ip_addresses` list back into the file, replacing its original contents with the revised allow list.

Python Syntax and Functions:

1. **`" ".join(ip_addresses)`**:
   - The `.join()` method converts the `ip_addresses` list back into a single string, with each IP address separated by a space. This ensures the format is consistent with the original file.
2. **`with open(import_file, "w") as file:`**:
   - Opens the file in write mode (`"w"`), which overwrites the file's contents with the new data.
3. **`file.write(ip_addresses)`**:
   - Writes the updated `ip_addresses` string to the file, replacing its previous contents.



```
        ip_addresses.remove(element)

# Convert `ip_addresses` back to a string so that it can be written into the text file

ip_addresses = " ".join(ip_addresses)

# Build `with` statement to rewrite the original file

with open(import_file, "w") as file:

  # Rewrite the file, replacing its contents with `ip_addresses`

  file = file.write(ip_addresses)
```

Summary

I created an algorithm that removes IP addresses identified in a `remove_list` variable from the `"allow_list.txt"` file of approved IP addresses. This algorithm involved opening the file, reading its contents as a string, and then converting this string into a list stored in the `ip_addresses` variable. I iterated through the IP addresses in the `remove_list` using a `for` loop. During each iteration, I evaluated whether the current element existed in the `ip_addresses` list. If it did, I applied the `.remove()` method to delete the element from `ip_addresses`. After this, I used the `.join()` method to convert the updated `ip_addresses` list back into a string. Finally, I rewrote the contents of the `"allow_list.txt"` file with the revised list of approved IP addresses, ensuring that restricted IPs were successfully removed. This process demonstrated my ability to manipulate data programmatically and maintain secure access control.