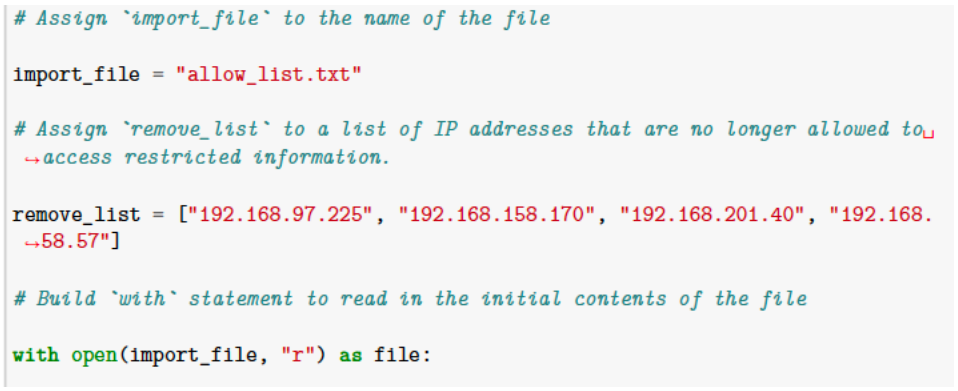
Algorithm for file updates in Python

Project description

[In this project, I worked as a security analyst to update an access control list stored in a text file. I used Python to parse the file, remove outdated IP addresses and save the updated list. The goal was to automate the process of maintaining accurate access records.]

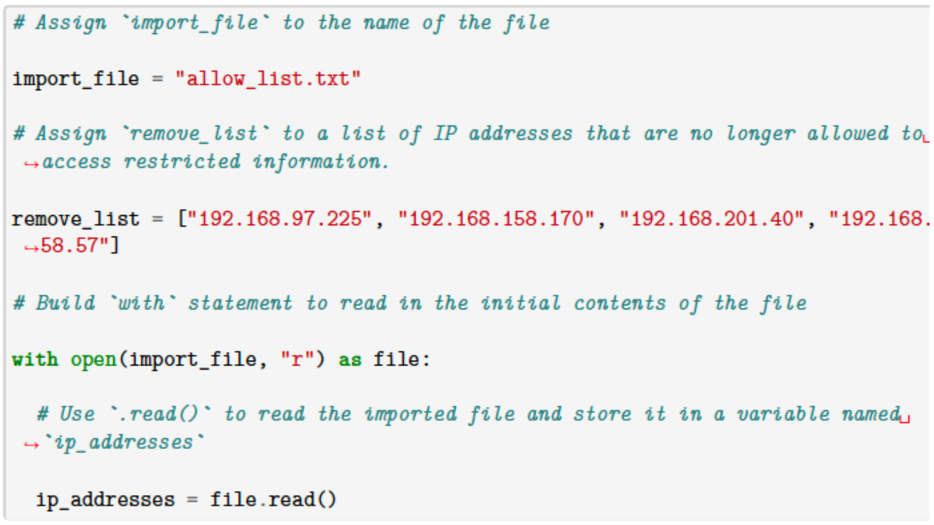
Open the file that contains the allow list



```
# 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"]

# Build `with` statement to read in the initial contents of the file

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

[
To open the file, I used the with statement and the open() function with the "r" mode for reading. This ensures the file is properly handled and closed afterward.
]

Read the file contents



```
# 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"]

# Build `with` statement to read in the initial contents of 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_addresses = file.read()
```

[I used the .read() method to load the entire contents of the file into a variable called ip_addresses. This converts the file content into a single string.]

Convert the string into 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)
```

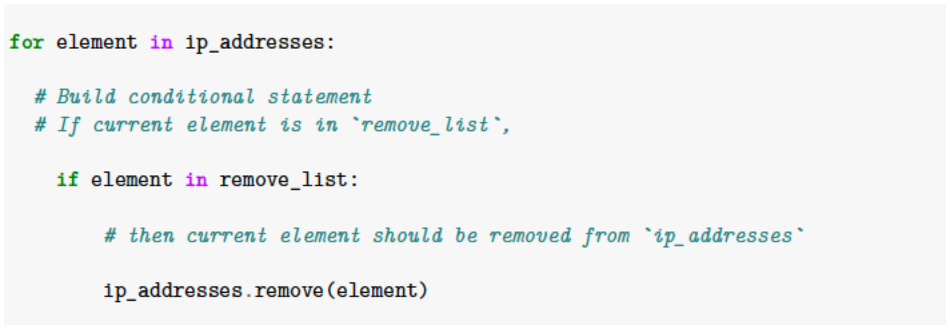
[To allow processing of individual IP addresses, I converted the string into a list using the .split() method, which separates the string by whitespace]

Iterate through the remove list

```
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)
```

[I created a for loop to iterate through each IP address in the remove_list, using element as the loop variable.]

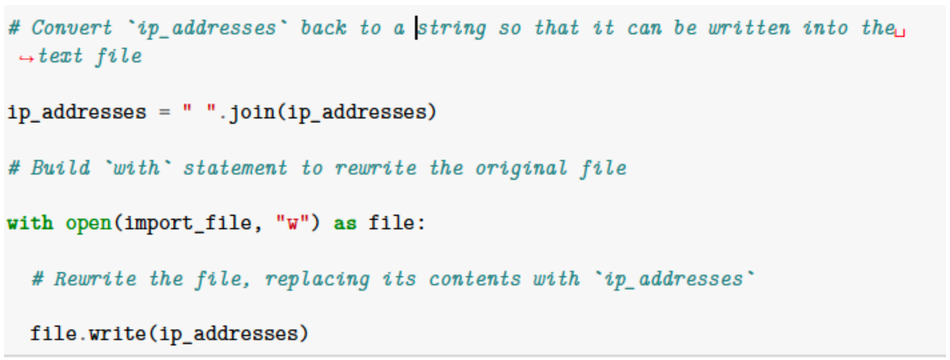
Remove IP addresses that are on the remove list



```
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)
```

[Inside the loop, I used an if statement to check if the IP should be removed. If so, I used the .remove() method on the list.]

Update the file with the revised list of IP addresses



```
# 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.write(ip_addresses)
```

[To rewrite the file with updated contents, I first used " ".join(ip_addresses) to convert the list back into a string. Then I opened the file in "w" mode and wrote the string back into it.]

Summary

[This algorithm uses Python to read and update a text file containing a list of IP addresses. The steps I followed were to read file contents with .read(), converting the string to a list using .split(), and removing specified entries with a loop and .remove(). The updated list is written back to the file using .join() and .write(). Encapsulating this logic in a function like update_file() enhances reusability and clarity. This method can be used in real-world scenarios to manage access control efficiently.]