Algorithm for file updates in Python

Project description

As a security professional at a healthcare organization, I manage access to a restricted subnetwork containing sensitive patient data. This access is controlled through an allow list of employee IP addresses stored in a file called `allow_list.txt`. Over time, access must be revoked for specific employees, and their IPs are added to a separate `remove_list`. To streamline this process, I created a Python algorithm that automatically removes the IPs in the remove list from the allow list file. This helps maintain up-to-date security controls with minimal manual effort.

Open the file that contains the allow list

To begin the process, I opened the `"allow_list.txt"` file and assigned its name as a string to the variable `import_file`.

Then, I used `with` a statement along with the `open()` function to open the file:

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

import_file = "allow_list.txt"
```

```
# First line of `with` statement

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

The `with` keyword ensures the file is closed adequately after accessing it. I passed `"r"` into `open()` to indicate the file should be read. I also used `as file` to store the opened file in a variable named `file` while working inside the `with` block.

Read the file contents

To read the contents of the `"allow_list.txt"` file, I used a `with` statement along with the `.read()` method. This ensures the file is opened correctly and closed, allowing me to store its contents in a variable. Here's the exact code I used:

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

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

  ip_addresses = file.read()
```

This code uses the `with` keyword to open the file assigned to the variable `import_file` in read mode, indicated by `"r"`. The `open()` function loads and assigns the file to the variable `file`. Inside the block, I used the `.read()` method on the `file` object to read all the file contents and store them as a single string in the variable `ip_addresses`.

Convert the string into a list

Since I needed to work with individual IP addresses, I converted the string into a list using the `.split()` method:

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

ip_addresses = ip_addresses.split()
```

By calling `ip_addresses.split()`, I broke the string into separate list elements based on whitespace. This allowed me to work with each IP address individually and perform list-based operations like removal. The result was reassigned back to the `ip_addresses` variable.

Iterate through the remove list

Next, I needed to go through each IP address in the `remove_list`. I used a `for` loop and named the loop variable `element` to represent each IP:

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

for element in ip_addresses:
```

The loop allows me to individually examine each item in the `ip_addresses` list to determine whether it should be removed.

Remove IP addresses that are on the remove list

To do this, I used a `for` loop to iterate through each IP address in the `ip_addresses` list. I added a conditional statement inside the loop that checked if the current element (i.e., the current IP address) was also in the `remove_list`. If the condition was true, the `.remove()` method was used to delete that IP from the `ip_addresses` 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)
```

The `if element in remove_list:` line checks whether the current IP address from the allow list also exists in the list of IPs that should no longer have access. If it does, `ip_addresses.remove(element)` removes it from the allow list.

Update the file with the revised list of IP addresses

After filtering out the removed IPs, I needed to convert the list back into a string so it could be written back to the file. I used the `.join()` method and chose a space `" "` as the separator:

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

ip_addresses = " ".join(ip_addresses)
```

Then, I used a second `with` statement to open the file in `"w"` mode, which stands for "write." This mode overwrites the existing content in the file. Inside this block, I used the `.write()` method to replace the file's contents with the updated list of 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)
```

Summary

The algorithm follows a structured process to update the allow list efficiently. It first reads the contents of the allow list file and converts them into a list format for easier manipulation. Next, it iterates through a predefined remove list, checking for any matching IP addresses within the allow list. If a match is found, the IP address is removed to ensure restricted access compliance. The updated list is then converted into a string and rewritten into the original file, ensuring that unauthorized users are promptly removed. This approach streamlines access control updates, enhancing security while minimizing human error.