Algorithm for file updates in Python

Project description

As a security professional at a healthcare company, I am responsible for managing access to sensitive patient records. To ensure secure access control, I developed a Python algorithm that automates the process of updating an allow list of IP addresses stored in `allow_list.txt`*. The algorithm compares the allow list with the remove list of unauthorized IP addresses and removes any matches, ensuring only authorized users retain access to restricted content.*

Open the file that contains the allow list

To open the file `"allow_list.txt"`, I first assigned its name as a string to the variable `import_file`. This makes the file name reusable and easy to update if needed.

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

import_file = "allow_list.txt"
```

Then, I used a `with` statement to open the file and while I worked with it inside the block, I used the variable `file` to reference it.

```
# First line of `with` statement

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

The `with` statement, combined with the `open()` function, is required in my algorithm for accessing the IP addresses stored in the allow list securely. This setup ensures the file is automatically closed after the code completes, preventing resource leaks. The line `with open(import_file, "r") as file:` includes two key parameters for the `open()` function: the file name (`import_file`) and the mode (`"r"`), which specifies that the file should be opened in read mode. Additionally, the `as` keyword assigns the file object to the variable `file`, making it simple and efficient to interact with the file's contents within the `with` block.

Read the file contents

To read the file contents, I used the `.read()` method to convert them into a string.

```
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 `open()` function with the `"r"` argument (read mode) to access the file within a `with` statement. Inside the block, I called the `.read()` method to convert the file's contents into a string, making it easy to process. I assigned the resulting string to the variable `ip_addresses` for further use.

Overall, this code efficiently reads the contents of `"allow_list.txt"` into a string format, which I later utilized to organize and extract data.

Convert the string into a list

To remove individual IP addresses from the allow list, I needed the IP addresses in a list format. So, I used the `.split()` method to convert the `ip_addresses` string into a list.

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

ip_addresses = ip_addresses.split()
```

The `.split()` method is applied directly to a string variable to transform its contents into a list. Its purpose in this context is to simplify the removal of specific IP addresses from the allow list. By default, `.split()` uses whitespace as the separator, which is ideal since the IP addresses in the `ip_addresses` string are separated by spaces. When this method is used, it processes the `ip_addresses` string, breaks it into individual addresses, and arranges them into a newly created list. The new list is saved back to the `ip_addresses` variable to make the next steps easier to manage.

Iterate through the remove list

Iterating through the IP addresses in the remove_list is a crucial step in my algorithm, achieved through the use of a `for` loop.

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

for element in ip_addresses:
```

In Python, the `for` loop is designed to execute specific code for each element in a sequence. It begins with the `for` keyword, followed by a loop variable (in this case, `element`), and the `in` keyword, which signals the loop to iterate through the sequence. For this algorithm, the sequence is `remove_list`, and during each iteration, the current value from `remove_list` is assigned to the loop variable `element`. This ensures each IP address is processed individually, enabling efficient removal.

Remove IP addresses that are on the remove list

To ensure that any IP address present in both `ip_addresses` and `remove_list` was removed from the allow list, I implemented a `for` loop with a conditional check.

```
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 conditional verified if the loop variable, `element`, existed in the `ip_addresses` list before applying `.remove()`. This step was necessary to prevent errors, as attempting to remove an element that isn't present in the list would raise an exception. Once the condition was met, I used the `.remove()` method on `ip_addresses`, passing `element` as its argument. This systematically removed each matching IP address from `ip_addresses`, ensuring the allow list was updated accurately.

Update the file with the revised list of IP addresses

As the final step in my algorithm, I needed to update the file containing the allow list with the revised IP addresses. First, I converted the `ip_addresses` list back into a single string using the `.join()` method.

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

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

This method merges all items from an iterable into a string, using a specified separator between each element. In this case, I chose `"\n"` as the separator so that each IP address would appear on a new line in the final string. This string was then ready to be written back to the file.

To update the file, I used a `with` statement alongside the `open()` function in write mode, specified with the argument `"w"`.

```
# 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)
```

This mode ensures that the file's previous contents are overwritten with the new data. Within the `with` block, I called the `.write()` method on the file object and passed the updated `ip_addresses` string as an argument. This action replaced the file's content with the revised allow list, ensuring that any removed IP addresses were no longer part of the list. By structuring the code this way, I ensured that the file was properly updated and securely closed after the operation.

Summary

I developed an algorithm to remove IP addresses from the `remove_list` variable in the "allow_list.txt" file of approved IPs. The first step was opening the file, reading its contents, and converting them into a string, which was then turned into a list stored in the `ip_addresses` variable. Next, I looped through the `remove_list` and checked if each IP address was in the `ip_addresses` list. If it was, I applied the `.remove()` method to remove it. Once that was

done, I used the `.join()` method to convert the updated list back into a string. Finally, I used the `.write()` method to overwrite the file with the revised list of IP addresses.