Update a File Through a Python Algorithm

Project description
In my workplace, we utilize a system that regulates access to confidential content by maintaining a predefined list of permitted IP addresses. To streamline the updating process for this list, I created an algorithm that automates modifications to the "allow_list.txt" file, specifically by removing IP addresses that should no longer have access.

Opening the file that contains the allow list
To initiate this process, I assigned the filename to a string variable named "import_file." Subsequently, I employed the "with" statement to open the file:

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

import_file = "allow_list.txt"
```

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

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

Within my algorithm, I utilized the "with" statement in combination with the ".open()" function to open the "allow_list.txt" file in read mode. This allowed me to access the IP addresses stored within the file. The "with" keyword ensures proper resource management by automatically closing the file after exiting the "with" statement. In the code snippet "with open(import_file, "r") as file:", the "open()" function takes two parameters. The first parameter specifies the file to import, and the second parameter indicates the intended operation. In this case, "r" signifies reading the file. The "as" keyword is used to assign the resulting file object to a variable named "file".

Reading the file contents
To read the contents of the file, I employed the ".read()" method to convert it 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()
```

When using the ".open()" function with the argument "r" for "read", I can call the ".read()" function within the "with" statement. The ".read()" method converts the file into a string, allowing me to read its contents. I applied the ".read()" method to the "file" variable defined in the "with" statement. The resulting string was assigned to a variable called "ip_addresses".

In summary, this code snippet reads the contents of the "allow_list.txt" file into a string format, enabling me to organize and extract data in my Python program.

Converting the string into a list

To facilitate the removal of individual IP addresses from the allow list, I needed 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()" function, appended to a string variable, converts the contents of a string into a list. This operation splits the text into list elements based on whitespace by default. In this algorithm, I applied the ".split()" function to the "ip_addresses" variable, which is a string containing IP addresses separated by whitespace. This transformed the string into a list of IP addresses, which I reassigned to the "ip_addresses" variable.

Iterating through the remove list

A crucial part of my algorithm involves iterating through the IP addresses listed in the "remove_list". To accomplish this, I incorporated a for loop:

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

for element in remove_list:
```

The for loop in Python allows for code repetition over a specified sequence. In the context of this algorithm, the purpose of the for loop is to execute specific code statements for each element in a sequence. The loop begins with the "for" keyword, followed by the loop variable "element" and the "in" keyword. "in" indicates that we will iterate through the "ip_addresses" sequence and assign each value to the loop variable "element".

Removing IP addresses from the allow list

My algorithm requires removing any IP address from the allow list, represented by "ip_addresses", that is also present in the "remove_list". Since there are no duplicate entries in "ip_addresses", I can achieve this using the following code:

```
for element in remove_list:

  # Create conditional statement to evaluate if `element` is in `ip_addresses`

    if element in ip_addresses:

      # use the `.remove()` method to remove
      # elements from `ip_addresses`

        ip_addresses.remove(element)
```

Within the for loop, I created a conditional statement to check if the loop variable "element" is present in the "ip_addresses" list. This check is necessary to avoid errors when applying the ".remove()" method to elements not found in "ip_addresses".

Within that conditional statement, I utilized the ".remove()" method on "ip_addresses" and passed the loop variable "element" as the argument. This ensured that each IP address in the "remove_list" was removed from "ip_addresses".

Updating the file with the revised list of IP addresses

As the final step in my algorithm, I needed to update the "allow_list.txt" file with the revised list of IP addresses. To achieve this, I first converted the list back into a 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)
```

The ".join()" method concatenates all items in an iterable into a string. In this algorithm, I employed the ".join()" method to create a string representation of the "ip_addresses" list. This string was then passed as an argument to the ".write()" method when writing to the "allow_list.txt" file. To separate each element with a new line, I used the string ("\n") as the separator.

I then used another "with" statement and the ".write()" method to update the file:

```
  # 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 time, I used the "w" argument with the "open()" function in the "with" statement, indicating the intention to write and replace the file's contents. When using the "w" argument, I can call the ".write()" function within the body of the "with" statement. The ".write()" function writes string data to the specified file, overwriting any existing content.

In this particular case, I wanted to write the updated allow list, represented by the "ip_addresses" variable, as a string to the "allow_list.txt" file. By appending the ".write()" function to the "file" object identified in

the "with" statement, I specified that the file's contents should be replaced with the data from the "ip_addresses" variable.

Summary

I developed an algorithm that removes IP addresses listed in the "remove_list" variable from the "allow_list.txt" file, which contains approved IP addresses. The algorithm involves opening the file, converting its contents into a readable string, and then converting that string into a list stored in the "ip_addresses" variable. I then iterated through the IP addresses in the "remove_list", checking for their presence in "ip_addresses". If found, I removed the respective element from "ip_addresses" using the ".remove()" method. Finally, I converted "ip_addresses" back into a string using the ".join()" method and updated the "allow_list.txt" file by overwriting its contents with the revised list of IP addresses.