Algorithm for file updates in Python

Project description

At my institution, access to restricted content is managed using an allow list of IP addresses stored in the "allow_list.txt" file. This file specifies the IP addresses that are allowed to access sensitive resources. Additionally, a exclusion list identifies IP addresses that should no longer be allowed access.

To simplify this process, I developed an algorithm that updates the "allow_list.txt" file by comparing it with the exclusion list. The algorithm detects IP addresses on the exclusion list and removes them from the permission list, so only authorized IPs are allowed to access restricted content.

Open the file that contains the allow list

```
import_file = "allow_list.txt"

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

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

To access the file containing the allow list, I used the **open()** function with the argument **"r"**, which specifies read mode. This allows the program to open the file without making any changes to its contents.

The **with** statement is utilized to handle the file safely. It ensures that the file is automatically closed after the block of code inside the **with** statement is executed. Within this block, the file is represented by the variable **file**, which can then be used to perform operations like reading its contents.

By using the **open()** function and the **with** statement, I ensured the file is securely opened and ready for further processing, such as reading or parsing the data it contains.

Read the file contents

I used the .read() method to read the file content and converted it to a string format.

```
import_file = "allow_list.txt"

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

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

  ip_addresses = file.read()

print(ip_addresses)
```

When using the .open() function, I can call the .read() method in the body of the with statement when the "r" argument (read mode) is included. The .read() method converts the content of the file into a string format and allows the content to be read.

In this code, the .read() method is applied to the file variable defined in the with statement. Then, the string output obtained from this method is assigned to the ip_addresses variable.

In summary, this code converts the contents of the "allow_list.txt" file into a string format, allowing this data to be used for editing and processing in my Python program.

Convert the string into a list

```
import_file = "allow_list.txt"

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

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

  ip_addresses = file.read()

ip_addresses = ip_addresses.split()

print(ip_addresses)
```

To remove individual IP addresses from the allow list, it is essential to convert the IP addresses from a string format into a list format. I achieved this using the **.split()** method.

After reading the contents of the file into the variable **ip_addresses** as a string, I applied the **.split()** method. This method splits the string into a list by default using whitespace (such as spaces or newlines) as the delimiter. Each IP address becomes an individual element in the resulting list.

The output of the **.split()** method is then reassigned to the **ip_addresses** variable, ensuring the data is in a format suitable for further operations like removing IPs listed in the remove list.

In summary, converting the string into a list allows for efficient processing and manipulation of the IP addresses in the allow list.

Iterate through the remove list

```
import_file = "allow_list.txt"

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

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

  ip_addresses = file.read()

ip_addresses = ip_addresses.split()

for element in ip_addresses:

    print(element)
```

To remove specific IP addresses from the **ip_addresses** list, we need to iterate through the **remove_list**, which contains the IP addresses that should be removed. This is accomplished using a **for loop**.

In this section, I set up the header of a **for loop** to iterate through each IP address in the **remove_list**. The loop variable is named **element**, which will hold the current IP address from the **remove_list** during each iteration.

Remove IP addresses that are on the remove list

```
import_file = "allow_list.txt"

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

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

  ip_addresses = file.read()

ip_addresses = ip_addresses.split()

for element in ip_addresses:

    if element in remove_list:

        ip_addresses.remove(element)

print(ip_addresses)
```

My algorithm needs to remove any IP address from the allow list, **ip_addresses**, that is also present in the **remove_list**. Since there are no duplicates in **ip_addresses**, I was able to use the following code to accomplish this:

First, within my **for loop**, I created a conditional that checked if the loop variable **element** existed in the **remove_list**. I did this because directly removing an element that was not found in **ip_addresses** would result in an error.

Then, inside that conditional, I applied the **.remove()** method to **ip_addresses**. I passed the loop variable **element** as the argument so that each IP address found in the **remove_list** would be removed from the **ip_addresses** list. This ensures that only authorized IPs remain in the allow list.

Update the file with the revised list of IP addresses

```
import_file = "allow_list.txt"

remove_list = ["192.168.97.225", "192.168.158.170", "192.168.201.40", "192.168.58.57"]
with open(import_file, "r") as file:

  ip_addresses = file.read()

ip_addresses = ip_addresses.split()
for element in ip_addresses:

    if element in remove_list:

        ip_addresses.remove(element)

ip_addresses = " ".join(ip_addresses)
with open(import_file, "w") as file:
  file.write(ip_addresses)
```

As a final step in my algorithm, I needed to update the allow list file with the revised list of IP addresses. To achieve this, I first needed to convert the list back into a string. I used the `.join()` method for this:

The `.join()` method combines all items in an iterable into a string. It is applied to a string containing characters that will separate the elements in the iterable once they are joined into a string. In this case, I used `" "` (a space) as the separator to concatenate the IP addresses in the `ip_addresses` list, ensuring they are placed in a single string.

Next, I used the `.join()` method to create a string from the `ip_addresses` list so I could pass it as an argument to the `.write()` method when updating the **"allow_list.txt"** file. The resulting string would replace the previous content of the file.

Then, I used another `with` statement and the `.write()` method to update the file:

This time, I used the second argument of **"w"** in the `open()` function in the `with` statement. This argument specifies that the file should be opened for writing, and any existing content will be overwritten. When using **"w"**, I can call the `.write()` method inside the `with` statement to write string data to the file.

In this case, I wanted to write the updated allow list as a string to the **"allow_list.txt"** file. This ensures that the restricted content will no longer be accessible to any IP addresses that were removed from the allow list.

Finally, I used the `.write()` function to replace the file's contents with the updated `ip_addresses` variable, ensuring the new list of allowed IP addresses was saved to the file.

Summary

I created an algorithm, this algorithm removes the IP addresses defined in the remove_list variable from the approved IP addresses in the "allow_list.txt" file. The algorithm involved opening the file, converting the data in the file into a readable string, and transforming that string to be stored in a list called ip_addresses. Then, I iterated over each IP address in the remove_list, checking if each item was present in the ip_addresses list. If the item is in the list, I remove it from the ip_addresses list using the .remove() method. After this, I used the .join() method to convert the ip_addresses list back to a string so that I could write the contents of the file to replace it with the updated list of IP addresses.