Algorithm for file updates in Python

Project description

My organization has an `"allow_list.txt"` file that specifies a list of IP addresses allowed into the network. They also maintain a list of IP addresses that should not be granted access. The content is restricted so that only IP addresses from the `"allow_list.txt"` file can access it. I created an algorithm that updates the `"allow_list.txt"` file by removing IP addresses that should not have access to the content.

Open the file that contains the allow list

I first assigned the file's name to a variable in python. The variable is named `import_file`.

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

import_file = "allow_list.txt"
```

I used a with statement to open the file, and since I only wanted to read its contents, I specified the parameter as `'r'`.

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

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

I used a `with` statement in my algorithm to open the file, as it manages external resources efficiently. The `with` statement does not require me to close the file after using it, which reduces the amount of code I need to write. Then, I used the `open()` function in read mode to open the `"allow_list.txt"` file and read its contents. This gave me access to all the content in the file (all the allowed IP addresses). The `open()` function takes two parameters: one that specifies the file and one that specifies the mode. I then stored the file as a local variable named `'file'`.

Read the file contents

In order to effectively read the file's contents, I used the .read() method. This method helped me convert the contents of the file to a string that I can parse later.

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

ip_addresses = file.read()
```

Since I used the `open()` function with the mode specified as **'r'** for "read," I was able to use the `read()` method within the `with` statement. The `.read()` method helped me retrieve the contents of the file and store them in a variable named `ip_addresses`.

Convert the string into a list

To remove or edit the IP addresses in the file, I need the contents to be in a list format. The contents are currently stored as a string, so I need to convert them into a list.

I used the `.split()` method to help me do this.

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

ip_addresses = ip_addresses.split()
```

The `.split()` method is called on the string value that contains the contents of the file. This effectively converts the contents of the string into a list. One advantage of converting the contents into a list is that I can easily remove IP addresses from the allowed list. I called the `.split()` function with no parameters (default), which splits the string by whitespace into list elements. In this code, the `.split()` function takes the characters stored in `ip_addresses`, which contains a string of IP addresses separated by whitespace, and converts it into a list, which is stored using the same variable (`ip_addresses`).

Iterate through the remove list

In order to remove IP addresses in the `remove_list`, I have to iterate through the list. I used a for loop to iterate through the `remove_list`:

```
# Build conditional statement
# If current element is in `remove_list`,

if element in remove_list:
```

The `for` loop is used to repeat a set of Python instructions a specific number of times. The `for` keyword at the beginning of the loop defines the start of the loop. It is followed by an element variable and the keyword `in`. The `in` keyword specifies that the `for` loop should iterate through a list. In this case, we want the `for` loop to iterate over the number of elements in

`remove_list`. Each time the iteration occurs, the specific IP address is temporarily stored in the element variable.

Remove IP addresses that are on the remove list

I need to remove any IP address from the allow list, stored in the variable named `ip_addresses,` that is also contained in `remove_list`. Since there were no duplicates in `ip_addresses`, I used 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)
```

Inside my `for` loop, I added a conditional statement that checks if the element (an IP address in the `remove_list`) is in the `ip_addresses` list (the list of allowed IP addresses). If it is, the algorithm removes the IP address from the `ip_addresses` list. This ensures that any IP address in the `remove_list` is not included in the approved `ip_addresses` list.

To remove the specific element, I used the `.remove()` method and passed the element as the argument. In conclusion, any IP address in the `remove_list` will not be in the `ip_addresses` list (approved list).

Update the file with the revised list of IP addresses

Finally, I need to update the allow list file with the new list of IP addresses. To do this, I converted `ip_addresses`, a list containing all allowed IP addresses, to 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 converts all the elements in a list into a string. In this case, it converts the list of IP addresses in the `ip_addresses` list into a string, with each IP address separated by a new line. The `.join()` method is called on `"\n"`, which specifies how the elements in

the list should be separated. I am converting everything in the `ip_addresses` list into a string so I can write it to the file using the `.write()` method.

I used another with statement with the write((‘w’) mode to revise 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)
```

These lines of code used a different argument (‘w’) with the `open()` function. This argument prepares the file to be modified and overwritten. When I am in write mode, I can use the `.write()` method. This method writes the specified string data to the specified file and overwrites any existing content.

My algorithm involved writing the updated string content in `ip_addresses` to the `“allow_list.txt”` file. This restricts access to any IP addresses that are not listed in the `“allow_list.txt”`” file. I called the `.write()` function on the file I wanted to overwrite, which in this case is `“allow_list.txt,”` and passed `ip_addresses` as an argument.

Summary

In this project, I created an algorithm that effectively removes all IP addresses in the `remove_list` variable from the `“allow_list.txt”` file. The `“allow_list.txt”` file contains all allowed IP addresses that can access the content. The algorithm opened the file, converted its contents to a string, and then converted the string to a list named `ip_addresses`. Then, I iterated through the elements (IP addresses) in `remove_list` and checked if each element was in the `ip_addresses` list. If the elements matched, I used the `.remove()` method to remove the specific element from the `ip_addresses` list. After this, I used the `.join()` function to convert `ip_addresses` back to a string. Finally, I wrote to the `“allow_list.txt”` file so it would contain the updated list of approved IP addresses.