Algorithm for file updates in Python

Project description

I must create an algorithm to parse a file and remove any IP addresses that are not in the allow list in order to update the list of employees who can access restricted content.

Open the file that contains the allow list

The first part of the algorithm requires opening the text file that I am working with. To do this, I assigned the file name to the variable **import_file** and the list of the IP addresses to be removed was stored in the variable **remove_list**. Underneath that is the first line of code required to actually open the file to be read. This code consists of **with open(import_file, 'r') as file:**. This line is telling our algorithm to **open** the **import_file** variable, and the **r** indicates to open it as read only. This prevents us from changing the contents of the file currently.

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

import_file = "allow_list.txt"

# Assign `remove_list` to a list of IP addresses that are no longer allowed to access restricted information.

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

# First line of `with` statement

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

Read the file contents

After defining our variables and starting our **open** statement, we have to create a new variable that will store the contents of our **import_file** variable. To do this, I included the line **ip_addresses = file.read()**. The **.read()** method will return the entire contents of the file in order to be used later on. Lastly I included a print statement to verify that the above code was working as intended.

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

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()

# Display `ip_addresses`

print(ip_addresses)
```

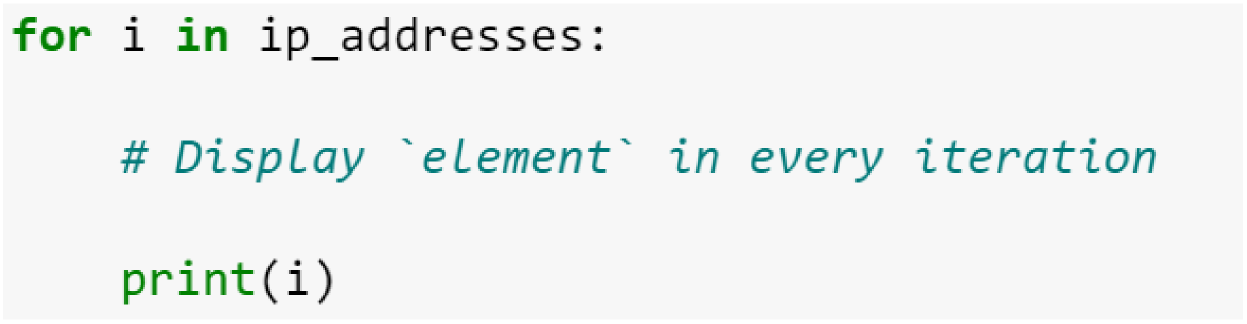
Convert the string into a list

After confirming the contents of the file was stored properly in the ip_addresses variable, my next step was to convert the contents from a string to a list. By converting the variable to a list, it will be easier to remove any desired IP addresses that are in the **remove_list**. To achieve this I used the **.split()** function. This function will take the data stored inside of the **ip_addresses** variable and separate it by each whitespace while converting it from a string to a list. (EX: a string “1 2 3” turns into a list [1, 2, 3])

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

ip_addresses = ip_addresses.split()

# Display `ip_addresses`

print(ip_addresses)
```

Iterate through the remove list

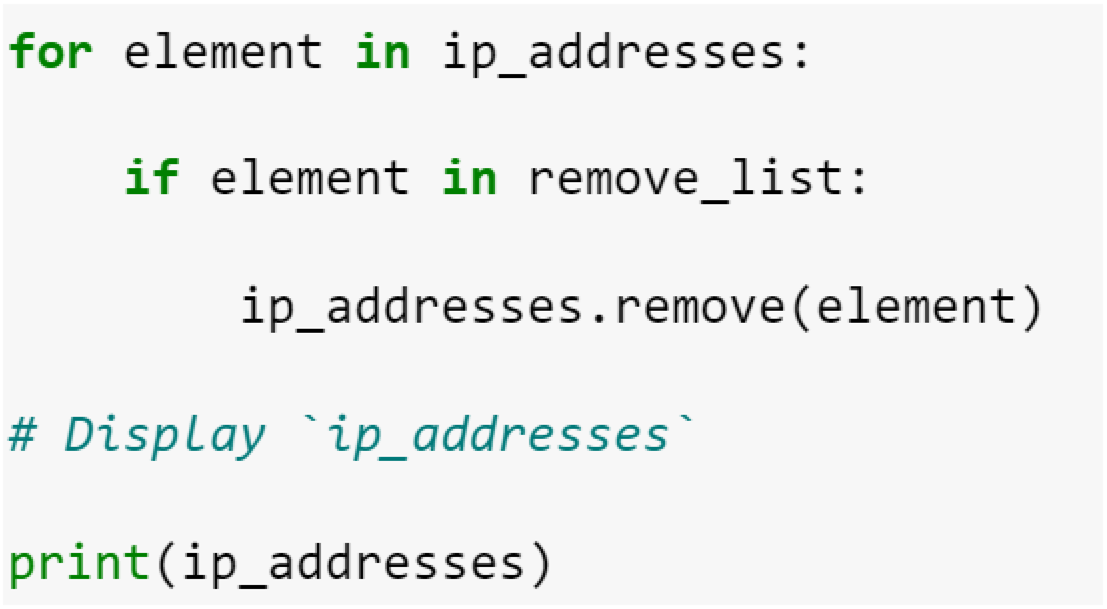
Now that we have our list of IP addresses, we can easily loop through and retrieve each IP address in the list. By using this **for loop**, we can also remove or edit any of the IP addresses inside as necessary. I started this **loop** by defining a variable **i** that will iterate through our new **ip_addresses** list. Within this for loop, I’m currently printing the variable **i** through each iteration to assure I am retrieving each IP address in the list correctly.



```python
for i in ip_addresses:

    # Display `element` in every iteration

    print(i)
```

Remove IP addresses that are on the remove list

After verifying that I was retrieving each IP address as intended, it is now safe to start removing the desired IP addresses. I changed the variable **i** to the variable **element** to enhance readability. Now, throughout each iteration in our for loop, we’re going to check if the current **element** can be found inside of the **remove_list** variable. I do this by adding the **if statement** showing “**if element in remove_list”**. If the **element** is not found inside of our **remove_list**, nothing will happen. However, if it is found in the list, that IP address should be removed entirely. This is achieved in the line **ip_address.remove(element)**. After the for loop is finished, I **print** the entire **ip_addresses** variable again to be sure that each IP address was removed correctly.



```python
for element in ip_addresses:

    if element in remove_list:

        ip_addresses.remove(element)

# Display `ip_addresses`

print(ip_addresses)
```

Update the file with the revised list of IP addresses

Now that I confirmed all of the desired IP addresses have been removed as intended, it's time to update our file with the new desired information. To do this, I have to convert the list from the previous steps back into a string of data. I start this out by **updating** the **ip_addresses** variable to the value of **" ".join(ip_addresses)**. This will take every index value from the **ip_addresses** list, and add it to a string separated by a space in a variable with the same name. After this new ip_address variable is established, I am able to start our new **open** statement similar to step 1. The only difference however, is instead of using an **r** for **read**, I used a **w** for **write**. By using **write**, I am able to change the contents of the specified file. In this case, I used **file.write(ip_addresses)** to **write** (or change) the previous contents, with the new contents of our **ip_addresses** variable that we passed through.

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

ip_addresses = " ".join(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

I successfully created an algorithm that removed all IP addresses included in the **remove_list** from the **allow_list.txt** file. This was done in order to remove access from restricted areas for employees who should not have access. This algorithm involved **opening, reading,** and **writing** to a text document, while using the **.remove()** and **.join()** methods to convert and change the contents of the original file.