Algorithm for file updates in Python

Project description

At my organization, access to restricted content is managed through an allow list of IP addresses stored in the "allow_list.txt" file. A separate remove list specifies IP addresses that should no longer be granted access. To streamline this process, I developed an algorithm that automatically updates the "allow_list.txt" file by removing any IP addresses found in the remove list.

Open the file that contains the allow list

For the first part of the algorithm, I opened the **"allow_list.txt"** file by assigning its name as a string to the `import_file` variable. I then used a `with` statement along with the `open()` function in read mode (`"r"`) to access the file. Opening the file in this way allowed me to retrieve the IP addresses stored in the allow list.

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

import_file = "allow_list.txt"
```

The `with` statement is used here because it automatically manages system resources by closing the file once the block is complete. In the code `with open(import_file, "r") as file:`, the first parameter specifies the file to open, and the second parameter indicates the mode of access—in this case, `"r"` for reading. The `as` keyword assigns the file object to the variable `file`, which I then used to work with the contents of the allow list.

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

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

Read the file contents

To read the file contents, I used the `.read()` method, which converts the file into a string.

When the `open()` function is called with the argument `"r"` for “read,” the `.read()` method can be used inside the `with` statement to retrieve the file’s contents. In my algorithm, I applied

`.read()` to the `file` variable created in the `with` statement and assigned its string output to the variable `ip_addresses`.

```
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()
```

In short, this step reads all the contents of **"allow_list.txt"** into a string, making it possible to later organize and extract the data in my Python program.

Convert the string into a list

To remove individual IP addresses from the allow list, I first needed to convert the data into a list format. I did this by using the `.split()` method on the `ip_addresses` string.

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

ip_addresses = ip_addresses.split()
```

The `.split()` method is called on a string and converts its contents into a list. By default, it separates the text by whitespace, creating individual list elements. In this case, the string of IP addresses stored in `ip_addresses` was split into a list, with each IP address becoming its own element. I then reassigned this list back to the `ip_addresses` variable so it could be more easily modified.

Iterate through the remove list

A key part of my algorithm involves iterating through the IP addresses in the `remove_list`. To accomplish this, I used a `for` loop.

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

for element in remove_list:
```

In Python, a `for` loop repeats a block of code for each element in a given sequence. In this algorithm, the loop begins with the `for` keyword, followed by the loop variable `element` and the keyword `in`. This structure tells Python to iterate through the sequence `ip_addresses`, assigning each value in the list to the variable `element` so that specific actions can be applied to each IP address.

Remove IP addresses that are on the remove list

My algorithm removes any IP address from the allow list (`ip_addresses`) that is also present in `remove_list`. Since there were no duplicates in `ip_addresses`, I was able to use a simple approach with a `for` loop and the `.remove()` method.

Within the loop, I first used a conditional statement to check whether the current loop variable `element` existed in `ip_addresses`. This check prevents errors that would occur if `.remove()` were called on an element not present in the list.

```
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)
```

If the condition was met, I applied `.remove()` to `ip_addresses`, passing the loop variable `element` as the argument. This removed each IP address found in `remove_list` from the allow list.

Update the file with the revised list of IP addresses

As the final step in my algorithm, I updated the allow list file with the revised list of IP addresses. First, I converted the list back into a string using the `.join()` method.

The `.join()` method combines all items in an iterable into a single string, using the string it is called on as a separator. In this algorithm, I applied `.join()` to the `ip_addresses` list, using

`"\n"` as the separator so that each IP address would appear on a new line. This produced a string suitable for writing back to the file.

Next, I used a `with` statement with `open()` in write mode (`"w"`) to update the file. The `"w"` argument indicates that the file should be opened for writing and that any existing content will be replaced. Inside the `with` block, I called `.write()` on the file object, passing in the updated `ip_addresses` string. This rewrote the **"allow_list.txt"** file, ensuring that IP addresses removed from the list no longer had access to the restricted content.

```
# 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 developed an algorithm to remove IP addresses listed in a `remove_list` from the **"allow_list.txt"** file of approved IPs. The algorithm first opens the file and reads its contents as a string, which is then converted into a list stored in the variable `ip_addresses`. I then iterate through each IP in `remove_list` and check if it exists in `ip_addresses`. If it does, the `.remove()` method is applied to delete that IP from the list. Finally, I use the `.join()` method to convert `ip_addresses` back into a string and overwrite the contents of **"allow_list.txt"** with the updated list, ensuring that removed IPs no longer have access.