Algorithm for file updates in Python

Project description

At my organization, access to restricted content is controlled with an allow list of IP addresses. The `"allow_list.txt"` file identifies these IP addresses. A separate remove list identifies IP addresses that should no longer have access to this content. I created an algorithm to automate updating the `"allow_list.txt"` file and remove these IP addresses that should no longer have access.

Open the file that contains that allow list

I need to tell the program the name of the file I want to work with. In this case, it's `"allow_list.txt"`. I store this name in a variable called `import_file` , Then I use a with statement, which is a special way to open files that makes sure they close properly when I'm done with them.

```
# I tell my program the name of the file where the allowed addresses are saved
import_file = "allow_list.txt"

# Then I use a special command that lets my program read what's inside this file
with open(import_file, "r") as file:
```

The `open()` function here is used with two pieces of information the name of the file and the `"r"` which stands for `"read"`. This means it's just going to look at the file, not change it. The `as` keyword is followed by `file` which is going to setting up `file` as a temporary name for `"allow_list.txt"` while it's open.

Read the file contents

To properly handle the file content, the program begins with opening the file in `'read'` mode. This mode is signaled by the `"r"` argument, which tells Python that to only look at the file, not change it.

```
# Opening the allow_list file in read mode and read its contents
with open(import_file, "r") as file:

    # Reading the content into ip_addresses as a string
    ip_addresses = file.read()
```

I applied the `.read()` method to the `file` variable, it means I took the action of reading and pointed it at the open file, Then, after capturing all that information in the form of text, I gave it a name, `ip_addresses`.

Convert the string into a list

After reading the file, we have all the IP addresses in one big string. To make it easier to work with each IP address, I needed to split this big string into a list. Each item in the list would be one IP address.

```
# Split the string of ip addresses into a list

ip_addresses = ip_addresses.split()
```

Using `.split()` on string of IP addresses chops it up at every space and gives back a bunch of smaller strings. Each of these smaller strings is an IP address, and they're all stored in a list called `ip_addresses`.

Iterate through the remove list

We have a second list, named `remove_list`, which has all the IP addresses we no longer want to allow. To check these against our `ip_addresses` list, we need to go through `remove_list` one by one and see if any are in `ip_adresses`.

```
for element in remove_list:
```

The word `for` tells Python to start looping. `Element` is the variable that will hold each IP address from the `remove_list` as it go through it.

Remove IP addresses that are on the remove list

To clean up from `"allow_list.txt"` by taking out any IP addresses that should no longer be allowed access. These unwanted IP addresses are listed in a separate collection known as `remove_list`.

```
# Looping through each IP address listed for removal
for element in remove_list:

    # Checking if the current IP address needs to be removed
    if element in ip_addresses:

        # Go ahead and remove the IP address from the allowed list
        ip_addresses.remove(element)
```

The `for` loop goes over each entry in `remove_list` and The `if` statement checks if the current `IP (element)` is in the `ip_address list` and If it is, `.remove(element)` gets rid of it. Each ip that matches gets removed, ensuring that only the IPs we want to keep are left in `ip_addresses`.

Update the file with the revised list of IP addresses

The final task of my algorithm was to save the updated list of IP addresses back into the `"allow_list.txt"` file. Before doing that, I needed to turn the list of individual IP addresses back into a single block of text, or a string. The `.join()` method helped us for this because it takes a list and sticks all the items together with a specified character in between each item. I chose to use a newline character (`"\n"`) which helps us each ip address would end up on its own line in the text file.

```
# Join the remaining ip addresses into a string, with each ip separated by a newline
ip_addresses = "\n".join(ip_addresses)
```

Next, I needed to open `"allow_list.txt"` again, but this time for writing, which is what `"w"` stands for in the `open()` function. This lets me replace the old content with the new, updated list.

```
# Opening the allow_list.txt file again, this time to change it, not just read it
with open(import_file, "w") as file:

    # Save the new list over the old one
    file.write(ip_addresses)
```

By using `"w"` I told the program that I want to overwrite everything in there with my new list. The `.write()` function will take that string I just made and printing it onto the file, replacing whatever was there before. So after this, the file only has the IP addresses we want to keep and none of the ones we removed.

Summary

So what I did was I made a Python algorithm effectively maintains the integrity of the `"allow_list.txt"` file by removing unauthorized IP addresses that can take a list of Ip addresses we no longer want to have access and remove them from our `"allow_list.txt."` This process opens the file, reads the addresses, checks them against the blocked list, takes out the ones we don't want, and then saves the updated list back to the file.