File Update Algorithm in Python

Update a File Through a Python Algorithm

Project Description:
At this fictitious organization, we use an *allow list* of IP addresses to control who can access restricted resources. This list is saved in a file called allow_list.txt. There's also a separate list of IPs that should no longer have access. Manually updating the file each time an IP needs to be removed isn't ideal, so I wrote a Python algorithm that automates this process. The script reads the current allow list, removes any IPs that appear on the *remove list*, and writes the cleaned-up list back to the file.

Step 1: Open the allow list file
To start, I assigned the filename to a variable called import_file. Then I used a with statement to open the file in read mode:

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

Using with here is important—it ensures the file gets closed properly afterward, which is cleaner and safer. The "r" argument means I'm opening the file to *read* it. Inside the with block, I can treat the file as a variable called file and use it to read the contents.

Step 2: Read the contents of the file
Once the file was open, I used the .read() method to grab everything inside and save it as a string:

```
ip_addresses = file.read()
```

Now, ip_addresses is just one long string that includes all the allowed IPs. I'll need to process it further before I can make any changes.

Step 3: Convert the string into a list
To work with individual IPs, I needed them in list format. That way, I could easily remove any that shouldn't be there. I used .split() to break the string into a list of IPs, separated by whitespace:

```
ip_addresses = ip_addresses.split()
```

This gave me a clean list of IPs to loop through and modify.

Step 4: Loop through and remove IPs
Next, I wrote a for loop to go through each IP in the list and check if it's on the remove_list. If it was, I removed it:

```
for element in ip_addresses:
    if element in remove_list:
        ip_addresses.remove(element)
```

This worked fine since there weren't any duplicates in the file. If an IP matched one from remove_list, it was removed right away.

Step 5: Write the updated list back to the file
After filtering out the unwanted IPs, I converted the list back into a string so it could be written back to the file. I used .join() to do this, putting each IP on a new line:

```
ip_addresses = "\n".join(ip_addresses)
```

Then I opened the same file again—this time in write mode—to replace its contents with the updated list:

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

Opening the file in "w" mode clears everything inside, so whatever I write afterward becomes the new content. This step finalized the update.

Step 6: Wrap it all up into a function
To make this script easier to reuse and maintain, I wrapped the entire process into a function called `update_file()`. This function takes two arguments: the filename of the allow list and the list of IPs to remove. Here's what the final function looks like:

```
def update_file(file_name, remove_list):
    # Step 1: Read the existing IP addresses from the file
    with open(file_name, "r") as file:
        ip_addresses = file.read()

    # Step 2: Convert the string to a list
    ip_addresses = ip_addresses.split()

    # Step 3: Remove any IPs that are in the remove_list
    for ip in remove_list:
        if ip in ip_addresses:
            ip_addresses.remove(ip)
```

```
# Step 4: Convert the list back into a string with each IP on a new line
updated_ips = "\n".join(ip_addresses)

# Step 5: Write the updated IP list back to the file
with open(file_name, "w") as file:
    file.write(updated_ips)
```

Now, anytime I need to update an allow list, I can simply call, for example:

```
update_file("allow_list.txt", ["192.168.1.100", "10.0.0.45"])
```

This makes the script flexible, clean, and easier to integrate into larger automation workflows or tools. By turning the logic into a function, I've made it much more scalable and practical for ongoing use.

Summary

In this project, I created a simple but effective Python script to keep an IP allow list up to date. It reads the file, turns the contents into a list, removes any IPs that no longer need access, and writes the updated list back to the file. I also wrapped this logic into a function called update_file() to make it reusable. This kind of automation is super helpful for keeping access control lists clean and accurate without having to manually edit text files every time an IP changes.