Algorithm for file updates in Python

Project description

My organization uses an allow list to control which IP addresses can access restricted content containing personal patient records. The "allow_list.txt" file stores approved IP addresses, and when employees no longer need access, their IP addresses get added to a remove list. I developed a Python algorithm that automatically reads the allow list file, checks for any IP addresses that appear on the remove list, removes those addresses, and updates the file with the revised list. This automation helps maintain security by ensuring only authorized employees can access sensitive patient information.

Open the file that contains the allow list

The first step in the algorithm opens the "allow_list.txt" file. I assigned the filename as a string to the `import_file`variable:

python

```
import_file = "allow_list.txt"
```

Then I used a with statement to open the file:

python

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

The `with` statement manages resources by automatically closing the file after the code block finishes executing. The `open()` function takes two parameters: the first specifies which file to open (`import_file`), and the second indicates the mode. Using `"r"` means I'm opening the file in read mode to access its contents. The `as` keyword assigns the file object to a variable named `file` that I can work with inside the with statement.

Read the file contents

To read the file contents, I used the `.read()` method to convert the file into a string:

python
```
with open(import_file, "r") as file:
    ip_addresses = file.read()
```

The `.read()` method converts the entire file contents into a string format. I applied this method to the `file` variable from the with statement and assigned the result to `ip_addresses`. This gives me a string containing all the IP addresses from the allow list, which I can then manipulate in Python.

Convert the string into a list

To remove individual IP addresses, I needed the data in list format. I used the `.split()` method to convert the string into a list:

python

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

The `.split()` method converts a string into a list by breaking it at each whitespace character. Since the IP addresses in the file are separated by whitespace (spaces or newlines), this method creates a list where each IP address becomes a separate element. I reassigned the result back to `ip_addresses`, so now it contains a list instead of a string. This list format makes it easier to iterate through and remove specific IP addresses.

Iterate through the remove list

Iterate through the remove list

The algorithm needs to check each IP address in the `remove_list` to see if it exists in the allow list. I used a for loop to iterate through the remove list:

python

```
for element in remove_list:
```

The `for` loop repeats code for each item in a sequence. The loop variable `element` takes on the value of each IP address in `remove_list` one at a time. The `in` keyword indicates that I'm iterating through the `remove_list` sequence. For each iteration, the code inside the loop will execute with `element` representing the current IP address being processed.

Remove IP addresses that are on the remove list

Inside the for loop, I added code to remove IP addresses from the allow list if they match addresses in the remove list:

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

The conditional `if element in ip_addresses:` checks whether the current IP address from the remove list exists in the allow list. This check prevents errors that would occur if I tried to remove an element that wasn't in the list. When the condition is true, the `.remove()` method deletes that IP address from `ip_addresses`. The `.remove()` method takes the element to be removed as its argument. This approach works because there are no duplicate IP addresses in the allow list, so each removal operation finds and deletes exactly one matching entry.

Update the file with the revised list of IP addresses

After removing the IP addresses, I needed to update the file with the revised list. First, I converted the list back into a string using the `.join()` method:

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

The `.join()` method combines all elements in a list into a single string. I applied it to the string `"\n"` (newline character), which tells Python to separate each IP address by placing it on a new line. This creates a properly formatted string where each IP address appears on its own line, matching the original file format.

Then I used another with statement and the `.write()` method to update the file:

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

This time I used `"w"` as the second argument to `open()`, which opens the file in write mode. Write mode replaces the entire contents of the file with new data. The `.write()` method takes the `ip_addresses` string and writes it to the file, completely replacing the old allow list with the updated version that no longer contains the removed IP addresses. This ensures that only authorized employees retain access to restricted content.

Summary

I created a Python algorithm that automates updating an IP address allow list by removing addresses that should no longer have access to restricted content. The algorithm opens the "allow_list.txt" file and reads its contents into a string, then converts that string into a list for easier manipulation. It iterates through each IP address in a remove list, checks if that address exists in the allow list, and removes it if found. After all removals are complete, the algorithm converts the updated list back into a string with each IP address on a new line, then writes this string back to the file to replace its contents. This automation ensures the access control list stays current without requiring manual file editing, reducing the chance of errors and maintaining security for sensitive patient information.