Algorithm for file updates in Python

Project description

I am a security professional working at a health care company. I'm required to regularly update the `"allow_list.txt"` file that identifies the employees who can access restricted content. The file contents are based on who needs access to personal patient records. Access restriction is based on IP addresses of employees. There is an allow list for IP addresses permitted to sign into the restricted subnetwork and a remove list that shows employees I need to remove from the allow list.

I need to create an algorithm that uses Python code to check whether the allow list contains any IP addresses identified on the remove list. If so, the algorithm should remove those IP addresses and restrict access to the personal patient records.

Open the file that contains the allow list

First, I opened the `"allow_list.txt"` file and assigned this file name as a string to the `import_file` variable. I then use a `with` statement to open the file.

```
import_file = "allow_list.txt"

# Assign `remove_list` to a list of IP addresses that no longer have access

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

# 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 as variable `ip_addresses`

  ip_addresses = file.read()
```

The `with` statement is used with the `.open()` function in read mode, as indicated by passing `"r"` the second parameter, to open the allow list file, or first parameter, for the purpose of reading it. This gives me access to the IP addresses stored in the allow list. The `as` keyword assigns a variable named `file` to store the output while working within the `with` statement.

Read the file contents

I used the `.read()` method (above) to convert `ip_addresses` into a string to read. I applied the `.read()` method to the `file` variable in the `with` statement and assigned the output to the variable `ip_addresses`. I then used the `print()` function to read the contents of the allowed IP addresses file.

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

  # Use `.read()` to read the imported file and as variable `ip_addresses`

  ip_addresses = file.read()

# Display `ip_addresses`

print(ip_addresses)
```

```
ip_address
192.168.25.60
192.168.205.12
192.168.97.225
192.168.6.9
192.168.52.90
192.168.158.170
192.168.90.124
192.168.186.176
192.168.133.188
192.168.203.198
192.168.201.40
192.168.218.219
192.168.52.37
192.168.156.224
192.168.60.153
192.168.58.57
192.168.69.116
```

Convert the string into a list

To remove the IP addresses in the `remove_list` from the allow list, I used the `.split()` method to convert the `ip_addresses` string into a list. The `.split()` function is called by appending it to a string variable.

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

   # Use `.read()` to read the imported file and as variable `ip_addresses`

  ip_addresses = file.read()

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

ip_addresses = ip_addresses.split()

# Display `ip_addresses`

print(ip_addresses)
```

Splitting `ip_addresses` into a list makes it easier to remove IP addresses. Without a call, the `.split()` function splits the text by whitespace. I stored the new list to the variable `ip_addresses`. Displaying the new output confirms it is now a list.

```
['ip_address', '192.168.25.60', '192.168.205.12', '192.168.97.225', '192.168.6.9', '192.168.52.90', '192.168.158.170', '192.
168.90.124', '192.168.186.176', '192.168.133.188', '192.168.203.198', '192.168.201.40', '192.168.218.219', '192.168.52.37',
'192.168.156.224', '192.168.60.153', '192.168.58.57', '192.168.69.116']
```

Iterate through the remove list

My algorithm then iterates through the IP addresses that are elements in the `remove_list` using a `for` loop, which repeats code for a specified sequence. The `for` keyword initiates the loop, followed by the loop variable `element` and the keyword `in`, which cycles through elements in the `ip_addresses` and iteratively assigns each value to `element`.

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

for element in ip_addresses:
```

Remove IP addresses that are on the remove list

My algorithm then removes IP addresses from the allow list, `ip_addresses`, if they are in the `remove_list`.

```
for element in ip_addresses:

  # Build conditional statement
  # If current element is in `remove_list`,

    if element in remove_list:

        # then current element should be removed from `ip_addresses`

        ip_addresses.remove(element)
```

I created a conditional in my `for` loop that evaluated whether the loop variable `element` was found in `ip_addresses`. Then I added `.remove()` to `ip_addresses`, passing the `element` as the argument to remove each IP address in `remove_list` from `ip_addresses`.

Update the file with the revised list of IP addresses

Finally, I updated the allow list file with the revised list of IP addresses. I used the `.join()` method to convert the list back into a string.

```
# Convert `ip_addresses` back to a string to be written into the text file

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

The `.join()` method created a string from the list `ip_addresses` to pass it in as an argument to the `.write()` method when writing to the file `"allow_list.txt"`. I used the string `(" ")` as the separator to instruct Python to add a space between each element. I used another `with` statement and the `.write()` method to update the file with the removed IP addresses taken out.

```
# 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)
```

I used `"w"` as the second argument of the `open()` function to indicate I wanted to write over the files contents. Within the body of the `with` statement, I called the `.write()` to write string data to `"allow_list.txt"` and replace existing content. This ensured restricted

content was no longer accessible to any removed IP addresses. I did this by appending the `.write()` function to `file` and passing in `ip_addresses` as the argument to specify the contents to replace the original content.

Summary

My algorithm removes IP addresses identified in a `remove_list` variable from the `"allow_list.txt"` file of approved IP addresses. To do this, I opened the file, converted it to a string, and then converted the string to a list stored in as `ip_addresses`. I then iterated through the IP addresses in `remove_list` and evaluated if each element was part of the `ip_addresses` list. If yes, I applied the `.remove()` method to it to delete the IP address from `ip_addresses`. I then used the `.join()` method to convert the `ip_addresses` back into a string to write over the contents of the `"allow_list.txt"` file with the revised list.