Algorithm for file updates in Python

Project description

I am a security professional working for a healthcare organization with restricted files. Permission to these files are granted by user IP address on the allow list. I have been tasked by my team to write an algorithm that uses Python code to check whether the allow list contains any IP addresses identified on the remove list. If so, you should remove those IP addresses from the file containing the allow list.

Open the file that contains the allow list

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

import_file = "allow_list.txt"
```

Screenshot above is the code is used to open the allow list then assigned it as a string to "import_file"

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

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

To open the file I wrote a code using the with statement.
In my algorithm, the `with` statement is used with the `.open()` function in read mode to open the allow list file for the purpose of reading it. The purpose of opening the file is to allow me to access the IP addresses stored in the allow list file. The `with` keyword will help manage the resources by closing the file after exiting the `with` statement. In the code `with open(import_file, "r") as file:`, the `open()` function has two parameters. The first identifies the file to import, and then the second indicates what I want to do with the file. In this case, `"r"` indicates that I want to read it. The code also uses the `as` keyword to assign a variable named `file`; `file` stores the output of the `.open()` function while I work within the `with` statement.

Read the file contents

```
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()

# Display `ip_addresses`

print(ip_addresses)
```

Screenshot above displays my code using the open() function with “r” for reading. By using the “r” function I can use this function to make my “with” statement. The read() function converts a file into a string and allows me to read it.

Convert the string into a list

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

ip_addresses = ip_addresses.split()

# Display `ip_addresses`

print(ip_addresses)
```

The screenshot above shows my code to convert data into a list. The `.split()` function is called by appending it to a string variable. It works by converting the contents of a string to a list. The purpose of splitting `ip_addresses` into a list is to make it easier to remove IP addresses from the allow list.

Iterate through the remove list

```
ip_addresses = ip_addresses.split()

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

for element in ip_addresses:

    # Display `element` in every iteration

    print(element)
```

The `for` keyword starts the `for` loop. It is followed by the loop variable `element` and the keyword `in`. The keyword `in` indicates to iterate through the sequence `ip_addresses` and assign each value to the loop variable `element`.

Remove IP addresses that are on 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)

# Display `ip_addresses`

print(ip_addresses)
```

Screenshot above shows my code to remove IP addresses that should not have access to the restricted file. To start this code, within my `for` loop, I created a conditional that evaluated whether or not the loop variable `element` was found in the `ip_addresses` list. I did this because applying `.remove()` to elements that were not found in ip_addresses would result in an error.

Then, within that conditional, I applied `.remove()` to `ip_addresses`. I passed in the loop variable `element` as the argument so that each IP address that was in the `remove_list` would be removed from `ip_addresses`.

Update the file with the revised list of IP addresses

```
    if element in remove_list:

        # then current element should be removed from `ip_addresses`

        ip_addresses.remove(element)

# Convert `ip_addresses` back to a string so that it can be written into the text file

ip_addresses = " \n".join(ip_addresses)

# 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)
```

The last step in my algorithm, I needed to update the allow list file with the revised list of IP addresses. To do so, I first needed to convert the list back into a string. I used the `.join()` method for this. The join function combines all items in an iterable into a string. I used the `.join()` method to create a string from the list `ip_addresses` so that I could pass it in as an argument to the `.write()` method when writing to the file `"allow_list.txt"`. I used the string `("\n")` as the separator to instruct Python to place each element on a new lin

Summary

For this task my responsibility was to create an algorithm that removes IP addresses from a “remove” list variable from the “allow” list file of approved IP addresses. To write this code, it involved opening the file, converting it to a string to be read, and then converting this string to a list stored in the variable IP_addresses. Then with the help of my coworkers I iterated through the IP addresses in “remove_list”. With each iteration, I evaluated if the element was part of the IP_addresses list. If it was, I applied the “.remove()” method to it to remove the element from IP_addresses. After this, I used the “.join()” method to convert the IP_addresses back into a string so that I could write over the contents of the “allow_list.txt” file with the revised list of IP addresses.