Algorithm for file updates in Python

Project description

My organization manages access to restricted content using an allow list stored in "allow_list.txt". To streamline updates, I built an algorithm that automatically removes IP addresses listed in a separate remove list, ensuring only authorized users retain access.

Open the file that contains the allow list

```
# Assign `import_file` to the name of the file
import_file = "allow_list.txt"
# Assign `remove_list` to a list of IP addresses that are no longer allowed to access restricted information.
remove_list = ["192.168.97.225", "192.168.158.170", "192.168.201.40", "192.168.58.57"]
# First line of `with` statement
with open(import_file, "r") as file:
```

The algorithm uses a with statement and the open() function in read mode ("r") to safely access an allow list file containing IP addresses. The with statement ensures the file is automatically closed after use. The open() function takes two arguments: the file name and the mode. The as keyword assigns the opened file to a variable (file) for use within the block.

Read the file contents

```
# Assign `import_file` to the name of the file
import_file = "allow_list.txt"
# Assign `remove_list` to a list of IP addresses that are no longer allowed to access restricted information.
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 store it in a variable named `ip_addresses`
  ip_addresses = file.read()
# Display `ip_addresses`
print(ip_addresses)
```

The code uses open() in read mode ("r") within a with statement to safely access the "allow_list.txt" file. Inside the block, the .read() method is applied to the file object to convert its contents into a string. This string is then stored in the variable ip_addresses, making it easier to process and extract data later in the program.

Convert the string into a list

```
# Assign `import_file` to the name of the file
import_file = "allow_list.txt"
# Assign `remove_list` to a list of IP addresses that are no longer allowed to access restricted information.
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 store it in a variable named `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)
```

The .split() method is used to convert the ip_addresses string—containing space-separated IPs—into a list. This makes it easier to manage and remove individual addresses from the allow list. By default, .split() separates the string by whitespace, and the resulting list is reassigned to the same variable for further use.

Iterate through the remove list

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

import_file = "allow_list.txt"

# Assign `remove_list` to a list of IP addresses that are no longer allowed to access restricted information.

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 store it in a variable named `ip_addresses`

  ip_addresses = file.read()

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

ip_addresses = ip_addresses.split()

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

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

    # Display `element` in every iteration

    print(ip_addresses)
```

A for loop in Python is used to repeat actions for each item in a sequence. In this case, it iterates through the ip_addresses list, assigning each IP to the loop variable element one at a time. This allows the algorithm to apply specific operations to every IP address in the list.

Remove IP addresses that are on the remove list

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

import_file = "allow_list.txt"

# Assign `remove_list` to a list of IP addresses that are no longer allowed to access restricted information.

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 store it in a variable named `ip_addresses`

  ip_addresses = file.read()

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

ip_addresses = ip_addresses.split()

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

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)
```

Inside the for loop, a conditional check ensures that each element exists in the ip_addresses list before attempting to remove it. This prevents errors from trying to remove non-existent items. If the condition is met, .remove() is called to delete that IP address from the list, effectively filtering out entries found in the remove_list.

Update the file with the revised list of IP addresses

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

import_file = "allow_list.txt"

# Assign `remove_list` to a list of IP addresses that are no longer allowed to access restricted information.

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 store it in a variable named `ip_addresses`

  ip_addresses = file.read()

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

ip_addresses = ip_addresses.split()

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

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)

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

ip_addresses = " ".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 .join() method is used to convert the ip_addresses list into a string, with each IP separated by a newline ("\n"). This formatted string is then written to "allow_list.txt" using the .write() method inside a with open(..., "w") block. The "w" mode overwrites the file's contents, ensuring that removed IPs are no longer present in the updated allow list.

Summary

I built an algorithm to clean up the "allow_list.txt" file by removing IP addresses listed in a remove_list variable. First, I opened the file and read its contents into a string, then split that string into a list called ip_addresses. I looped through each item in remove_list, checked if it existed in ip_addresses, and removed it if found. Once the list was updated, I used .join() to convert it back into a string—placing each IP on a new line—and overwrote the original file with the cleaned-up version.