Algorithm for file updates in Python

Project description

In my organization, access to restricted content is managed through an allowed list of specified IP addresses. This `allow_list.txt` file contains the identification of these permitted IP addresses.There is also a separate remove list that is maintained in order to identify the IP addresses no longer authorized to access the content. To do this, I developed an algorithm that automates the process of updating the `allow_list.txt` file, thus making sure that IP addresses no longer requiring access to the content will be removed.

Open the file that contains the allow list

To open the file `allow_list.txt`, I first assigned the name as a string to the `import_file` variable:

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

import_file = "allow_list.txt"
```

Then, I used a `with` statement to open the file:

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

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

My algorithm uses the `with` statement in conjunction with the `.open()` function in read mode in order to open the allow list file and be able to read it. This output enables me to access the IP addresses stored in the file. The use of the `with` keyword is also beneficial for resource management because it will automatically close the file upon exiting the `with` statement. In the code snippet `open(import_file, "r") as file:`, the open() function takes two parameters: the file to import and the intended operation on the file. In this instance, `"r"` signifies the operation that I intend to read the file. When it comes to the `as` keyword in this code snippet, it is used for assigning the file to be read to the variable named `file`, thus storing the output of the `.open()` function while I operate in the `with` statement.

Read the file contents

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

```
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()
```

When utilizing the `.open()` function with the argument `"r"` (read), I can invoke the `.read()` function within the `with` statement. The `.read()` method transforms the file into a string, thus allowing me to read what it says. I also used the `.read()` method on the `file` variable that was specified in the `with` statement, and then I assigned the resulting string output to the variable `ip_addresses`.

In other words, this code reads the `allow_list.txt` file contents as it converts them into a string format. This string format can then be used to organize and extract data in this Python program.

Convert the string into a list

In order to make it easier to remove IP addresses from the allow list, I will split them into an actual list instead of keeping them as a string of IP addresses. The `.split()` method is used for converting a string into a list, so I used it to convert the `ip_addresses` string using the following code:

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

ip_addresses = ip_addresses.split()
```

The function `.split()` works is by automatically dividing the inputted text into list elements based on the presence of whitespace. In this algorithm, the `.split()` function processes the data from the `ip_addresses` variable by converting the string into a list by splitting the addresses based on the whitespaces between them and then reassigning the resulting list back to the same variable.

Iterate through the remove list

To iterate through the elements of this list called `remove_list`, I used a `for` loop in my algorithm:

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

for element in remove_list:
```

The purpose of a Python `for` loop is to iterate over a defined sequence, thus allowing for the application of specific code statements to each element in that sequence. The `for` loop begins with the keyword `for`, followed by the loop variable `element` and the `in` keyword. The `in` keyword signifies the iteration through the sequence `ip_addresses`, assigning each value to the loop variable `element`.

Remove IP addresses that are on the remove list

This algorithm removes any IP address from the allow list, `ip_addresses`, that is also contained in `remove_list`. In this case there are no duplicates in `ip_addresses`, the following code accounts for it:

```
for element in remove_list:

  # Create conditional statement to evaluate if `element` is in `ip_addresses`

    if element in ip_addresses:

      # use the `.remove()` method to remove
      # elements from `ip_addresses`

        ip_addresses.remove(element)
```

First, I established a condition in my `for` loop that would detect the presence of the loop variable `element` within the `ip_addresses` list. This precautionary step was taken to avoid errors when applying `.remove()` to elements that weren't found in `ip_addresses`. Subsequently, within that conditional block, I executed the `.remove()` operation on `ip_addresses`, using the loop variable `element` as the argument. This ensured the removal of each IP address listed in the `remove_list` from the `ip_addresses` list.

Update the file with the revised list of IP addresses

To update the file with the revised version. I used the `.join()` method to revert the list `ip_addresses` back into a string so that I could input it to the `.write()` method when writing to the file `allow_list.txt`. I used the string `("\n")` as a separate way to instruct Python to position each element onto a new line.

```
# 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)
```

This time in my `with` statement, I used `"w"` with the `open()` function. This `"w"` argument indicates that I want to open a file to write over its existing contents and invokes the `.write()` function in the body of the `with` statement. This function will facilitate the writing of string data to the designated file and replace any pre-existing file content.

In this case, doing so is beneficial since we can update what IP addresses are allowed and remove the old contents.

Summary

I devised an algorithm to eliminate IP addresses that are in the `remove_list` from the `allow_list.txt` file, which contains approved IP addresses. This process consisted of opening the file, converting it into a string for reading purposes, and subsequently transforming this string into a list stored in the variable `ip_addresses`. Subsequently, I iterated through the IP addresses in the `remove_list`. During each iteration, I assessed whether the element was present in the `ip_addresses` list. If so, I employed the `.remove()` method to eliminate the element from `ip_addresses`. Following this, I used the `.join()` method to revert `ip_addresses` to a string, thus allowing me to overwrite the contents of the `allow_list.txt` file with the updated list of IP addresses.