Update a file through a Python algorithm

Project description

An allow list of IP addresses is used at my organization to limit access to content that is restricted. They are listed in the `"allow_list.txt"` file. IP addresses that should no longer have access to this content are listed on a separate remove list. To automatically update the `"allow_list.txt"` file and delete certain IP addresses that shouldn't have access, I developed an algorithm.

Open the file that contains the allow list

I started by opening the `"allow_list.txt"` file for the algorithm's first step. I first gave the `import_file` variable this file name as a string:

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

import_file = "allow_list.txt"
```

I then opened the file with a `with` statement:

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

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

In my algorithm, the allow list file is opened in read mode using the `.open()` function and the `with` statement to read it. My access to the IP addresses kept in the allow list file will be made possible by accessing the file. By shutting the file when the with statement is finished, the `with` keyword will aid in resource management. The `open()` function contains two parameters in the code with `open(import_file, "r") as file:` parameter. First, I specify the file to import, and then I specify what I want to do with that file in the second. The letter `"r"` here denotes my desire to read it. As I work inside the `with` statement, the code also uses the `as` keyword to assign a variable called `file`, which stores the results of the `.open()` function. I specify the `file` to import first, and then in the second I specify what I want to do with that file. I want to read it, as indicated by the letter `"r"` here. The code also uses the `as` keyword to assign a variable called `file`, which stores the output of the `.open()` function, while I work inside the `with` line.

Read the file contents

I transformed the contents of the file into a string using the `.read()` method so that I could read it.

```
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()
```

I can use the `.read()` function in the with statement's body when utilizing an `.open()` method that has the argument `"r"` for "read." I can read the file by converting it to a string using the `.read()` method. The file variable mentioned in the `with` statement was subjected to the `.read()` method. Then I set the variable `ip_addresses` to contain the textual output of this procedure.

In short, this method converts the `"allow_list.txt"` file's contents into a string format so that I may utilize the string in my Python application to organize and retrieve data.

Convert the string into a list

I required it to be in list format in order to remove specific IP addresses from the allow list. I therefore split the `ip_addresses` string into a list using the `.split()` method:

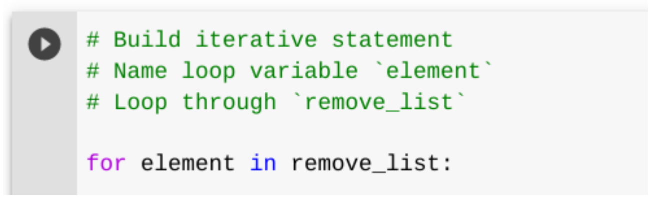
```
# Use `.split()` to convert `ip_addresses` from a string to a list

ip_addresses = ip_addresses.split()
```

You can use a string variable to append the `.split()` function to. It operates by turning a string's contents into a list. To make it simpler to delete IP addresses from the allow list, `ip_addresses` has been divided into a list. The `.split()` function divides text into list elements by default based on whitespace. In this algorithm, the data contained in the variable `ip_addresses`, which is a string of IP addresses, each separated by a whitespace, is converted into a list of IP addresses by the `.split()` function. I reassigned this list back to the `ip_addresses` variable to store it.

Iterate through the remove list

Iterating through the IP addresses that are included in the `remove_list` is a crucial component of my approach. In order to accomplish this, I used a `for` loop:



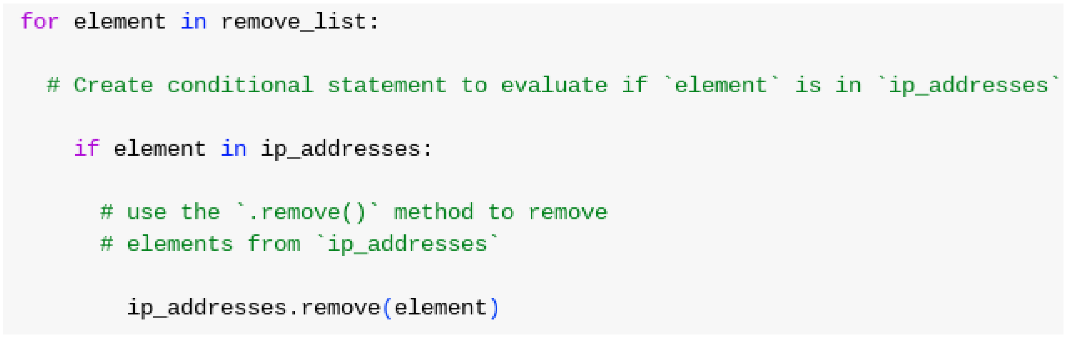
```
# Build iterative statement
# Name loop variable `element`
# Loop through `remove_list`

for element in remove_list:
```

Python's `for` loop repeats code for a given sequence. In a Python method like this, the `for` loop's main function is to apply particular code instructions to every element in a series. The `for` loop is initiated with the keyword. The loop variable `element` and the term "in" are next. The term in means to iterate through the list of `ip_addresses` and assign each value to a different element of the loop.

Remove IP addresses that are on the remove list

Any IP address in the allow list, `ip_addresses`, that is also in the `remove_list` must be removed according to my method. I was able to use the following code to perform this because `ip_addresses` did not contain any duplicates:



```
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 added a conditional to my `for` loop that checked to see if the loop variable `element` was present in the `ip_addresses` list. I did this because if I tried to apply `.remove()` to components that weren't in ip_addresses, it would fail.

I then applied the `.remove()` function to `ip_addresses` inside of that conditional. To ensure that each IP address in the `remove_list` was removed from `ip_addresses`, I passed in the loop variable `element` as an argument.

Update the file with the revised list of IP addresses

I had to update the allow list file with the updated list of IP addresses as the last step in my method. I have to first change the list back into a string in order to achieve that. For this, I employed the `.join()` method:

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

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

All of the entries in an iterable are combined into a string using the `.join()` method. When the iterable elements are merged into a string, the `.join()` method is applied to a string holding the characters that will divide them. To write to the file `"allow_list.txt"` using this algorithm, I created a string from the list of`ip_addresses` using the `.join()` method and then passed it as an argument to the `.write()` function. I told Python to start each element on a new line by using the separator string`("\n")`.

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

```
  # 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, I added a second argument of `"w"` to my `with` statement's use of the `open()` method. With this parameter, I'm trying to open a file so I can change its contents. I can use this argument `"w"` to invoke the`.write()` function in the `with` statement's main body. The `.write()` function replaces any existing file content and writes string data to the specified file.

In this instance, I wanted to add a string with the updated allow list to the`"allow_list.txt"` file. Any IP addresses that were deleted from the allow list will no longer have access to the restricted material. I added the `.write()` function to the file object `file` that I specified in the `with` statement in order to overwrite the file. I used the `ip_addresses` variable as a parameter to indicate that the data in this variable should be used in place of the contents of the file mentioned in the `with` statement.

Summary

I developed an algorithm that eliminates IP addresses from the `"allow_list.txt"` file of authorized IP addresses that are listed in a `remove_list` variable. Opening the file, transforming it to a string that could be read, and then turning that string into a list that was kept in the variable `ip_addresses` were all steps in this procedure. The IP addresses in `remove_list` were then iterated through. I determined whether an entry was a member of the `ip_addresses` list after each iteration. If it was, I removed the element from `ip_addresses` by applying the `.remove()` method to it. I then used the `.join()` function to turn the IP addresses back into strings so that I could replace the existing IP address list in the `"allow_list.txt"` file.