

Update a file through a Python algorithm

Project description

An allow list of IP addresses is used at my company to manage access to content that is forbidden. These IP addresses are identified in the "allow_list.txt" file. The IP addresses that ought to be blocked from accessing this content are listed in a separate delete list. I developed an algorithm to automatically remove these IP addresses that shouldn't be able to access the "allow_list.txt" file and update it.

Open the file that contains the allow list

First, I opened the "allow_list.txt" file to begin the algorithm. Initially, I put this file name in the `import_file` variable as a string:

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

import_file = "allow_list.txt"
```

Next, 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:
```

The allow list file is opened for reading in my algorithm by using the `with` statement and the `read-only.open()` function. I'm accessing the file so I may access the IP addresses that are kept in the allow list file. By exiting the `with` statement and closing the file, the `with` keyword will aid in resource management. Two parameters are passed to the `open()` function in the code that uses `open(import_file, "r")` as `file`. The first specifies which file has to be imported, and the second tells me what I want to do with it. "r" in this instance denotes my want to read it. Additionally, the code assigns a variable called `file` using the `as` keyword; `file` holds the result of the `.open()` method while I'm working inside the `with` statement.

Read the file contents

I converted the file's contents into a string using the `.read()` method in order to 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 call the `.read()` method in the body of the `with` statement when I use an `.open()` function that takes the argument "r" for "read." I can read the `file` by converting it to a string using the `.read()` method. I used the `with` statement's `file` variable to apply the `.read()` method. I then set the variable `ip_addresses` to contain the textual output of this procedure.

To summarize, this method reads the data from the `"allow_list.txt"` file into a format that can be used as a string later on in my Python application for organizing and extracting data.

Convert the string into a list

I required the list format in order to remove certain `ip_addresses` from the allow list. As a result, I then turned the string representing the `ip_addresses` into a list using the `.split()` method:


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

ip_addresses = ip_addresses.split()
```

By attaching it to a string variable, the `.split()` method is called. The way it operates is by turning a string's contents into a list. It will be simpler to delete IP addresses from the allow list if `ip_addresses` is divided into a list. The `.split()` function divides the text into list elements by default, using whitespace. In this algorithm, a string of IP addresses separated by whitespace is saved in the variable `ip_addresses`. The string is sent into the `.split()` function, which turns it into a list of IP addresses. I put this list back into the `ip_addresses` variable to store it.

Iterate through the remove list

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

```
 # 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 such as this one, the `for` loop's main goal is to apply particular code instructions to each element in a series. The `for` loop is started with the `for` keyword. The keyword `in` and the loop variable `element` come next. The keyword `in` instructs the loop variable `element` to assign each value as it iterates through the series of `ip_addresses`.

Remove IP addresses that are on the remove list

Any IP address that appears in `remove_list` and the allow list, `ip_addresses`, must be removed according to my method. I could use the following code to accomplish 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)
```

I started by adding a conditional to my `for` loop to determine whether the loop variable `element` could be found in the `ip_addresses` array. I took this action since `.remove()` would fail when used on components that weren't located in `ip_addresses`.

Then I used `.remove()` on `ip_addresses` inside that conditional. In order to remove every IP address that was in the `remove_list` from `ip_addresses`, I entered the loop variable `element` as the input.

Update the file with the revised list of IP addresses

The allow list file required to be updated with the updated list of IP addresses as the final step in my method. In order to accomplish this, I had to first turn the list back into a string. To accomplish this, I utilized 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)
```

The `.join()` method combines all items in an iterable into a string. The `.join()` method is applied to a string containing characters that will separate the elements in the iterable once joined into a string. In this algorithm, 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 line.

A string is created by joining every element in an iterable using the `.join()` method. To connect an iterable into a string, use the `.join()` method on a string of characters that will divide the elements. The `.join()` method in this algorithm was utilized to extract a string from the list of `ip_addresses` so that it could be passed as input to the `.write()` method for writing to the `"allow_list.txt"` file. For each element, I told Python to start on a new line by using the string `("\\n")` as the separator.

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 used the `open()` method in my `with` statement with a second argument of `"w"`. I want to open a file so I can write over its contents, as this argument suggests. I can invoke the `.write()` function in the `with` statement's body by utilizing this parameter, `"w"`. The `.write()` function replaces any existing file content and writes string data to a given file.

I wanted to write the revised allow list as a string to the `"allow_list.txt"` file in this particular instance. This will prevent any IP addresses that were taken off the allow list from

having access to the restricted material. The `.write()` function was inserted to the file object `file` that I had specified in the `with` statement in order to rewrite the file. To indicate that the data in this variable should be used in place of the contents of the file mentioned in the `with` statement, I supplied the `ip_addresses` variable as the argument.

Summary

I developed an algorithm that eliminates IP addresses from the "allow_list.txt" file of permitted IP addresses that are found in a `remove_list` variable. This algorithm read the file, converted it to a string that could be read, and then turned the string into a list that was kept in the `ip_addresses` variable. After that, I went through each IP address in `remove_list` one by one. I determined whether the element was included in the list of `ip_addresses` with each iteration. If so, I removed the element from `ip_addresses` using the `.remove()` method. Subsequently, I employed the `.join()` function to transform the IP addresses back into a string, enabling me to append the updated list of IP addresses to the "allow_list.txt" file's contents.