Algorithm for file updates in Python

Project description

In my company, restricted content can only be accessed by certain IP addresses. This list of allowed addresses is stored in a text file called `"allow_list.txt"`. On the other hand, the variable `remove_list` contains a series of IP addresses that need to be removed from the previous file. As a security analyst, I have created an algorithm that automatically removes those IP addresses from the `"allow_list.txt"` file and updates it. The `remove _list` contains:

- 192.168.97.225
- 192.168.158.170
- 192.168.201.40
- 192.168.58.57

Open the file that contains the allow list

In the first part of the algorithm, I opened the `"allow_list.txt"` file. You can see that I assigned this file as a string to the variable `import_file`. On the other hand, the list of IPs to be removed is assigned to the variable `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"]
```

Secondly, I opened the `import_file` using the `with` statement.

```
# First line of `with` statement

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

In my algorithm, you can see that I used the with statement with the `open()` function, specifying the reading of the file. The `with` statement ensures that all resources are managed and that, once the instructions are completed, the file is automatically closed. The `open()` function consists of two parameters. The first indicates the file to be imported, and the second indicates what you want to do with it. By specifying an `'r'`, the file will be read. This is followed by the keyword `as`, which assigns the output of the `open()` function to the variable file while working within the `with` statement.

Read the file contents

To read the contents of the file, the `.read()` method is used, which transforms the content into a string.

```
# 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)
```

As you can see in the code, by including the argument `'r'` in the `open()` function, it allows the use of the `.read()` method within the body of the `with` statement. This `.read()` method is applied to the variable `file`, which stores the content of `import_file`. Finally, the output string of this method is assigned to the variable `ip_addresses`. In summary, this code reads the contents of `"allow_list.txt"`, converts it into a string format, and then assigns it to a variable, allowing the data management and extraction from this string.

Convert the string into a list

The next step of the code is responsible for transforming the `ip_addresses` string into a list using the `.split()` method, in order to easily remove the prohibited IPs from the allowed list.

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

ip_addresses = ip_addresses.split()### YOUR CODE HERE ###

# Display `ip_addresses`

print(ip_addresses)
```

```
['ip_address', '192.168.25.60', '192.168.205.12', '192.168.97.225', '192.168.6.9', '192.168.52.90', '192.168.158.170', '192.16
8.90.124', '192.168.186.176', '192.168.133.188', '192.168.203.198', '192.168.201.40', '192.168.218.219', '192.168.52.37', '192.
168.156.224', '192.168.60.153', '192.168.58.57', '192.168.69.116']
```

This function is used with a string-formatted variable, which it converts into a list. By default, this function separates each element of a list based on the whitespace in the string. Therefore, this algorithm separates each IP in the `ip_addresses` variable according to the whitespace and converts them into a list of IP addresses. Finally, the updated variable is reassigned to `ip_addresses`.

Iterate through the remove list

One of the key parts of my algorithm involves iterating through the IPs contained in the list of the variable `remove_list`. For this, a `for` loop is used.

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

for element in remove_list:

    # Display `element` in every iteration

    print(element)
```

```
192.168.97.225
192.168.158.170
192.168.201.40
192.168.58.57
```

This loop repeats the instructions indicated in the code for each of the elements in a variable. First, the keyword `for` is used to start the loop. Next, the variable `element` stores each value of the variable. Finally, the keyword `in` indicates that the iteration will go through the `remove_list` list, assigning each of the values to the variable `element`.

Remove IP addresses that are on the remove list

The most important part of my algorithm involves removing IP addresses from the allowed list, `ip_addresses`, that also belong to the `remove_list`. Since there are no duplicate IP addresses in `ip_addresses`, a `for` loop could be used.

```
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)
```

```
['ip_address', '192.168.25.60', '192.168.205.12', '192.168.6.9', '192.168.52.90', '192.168.90.124', '192.168.186.176', '192.16
8.133.188', '192.168.203.198', '192.168.218.219', '192.168.52.37', '192.168.156.224', '192.168.60.153', '192.168.69.116']
```

First, a `for` loop is created to iterate over the elements of `ip_addresses`. Within this loop, I created a conditional with the `remove_list` to evaluate if an IP address (`element`) from `ip_addresses` is also found in `remove_list`. Finally, if it is present in both lists, the `.remove()` method is applied to `ip_addresses`. The loop variable `element` is used to assign each IP address, and if they match, it is deleted from `ip_addresses.`

Update the file with the revised list of IP addresses

As the final step, the initial text file of allowed IPs, `"allow_list.txt"`, must be updated with the new IP addresses. To do this, the updated list must first be converted to a string using 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)
```

This method combines all the iterable elements into a string. The combination is performed by specifying the element that will separate each of the values in the string. In this case, the `.join()` method creates a string from the `ip_addresses` list so that it can be used as an argument in the `.write()` method to update the `"allow_list.txt"` file. I used the string `"\n"` (newline) as a separator so that each element is on a new line.

Next, I used another `with` statement and the `.write()` method to write the content of `ip_addresses` to the `"allow_list.txt"` 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)
print(ip_addresses)
```

```
ip_address
192.168.25.60
192.168.205.12
192.168.6.9
192.168.52.90
192.168.90.124
192.168.186.176
192.168.133.188
192.168.203.198
192.168.218.219
192.168.52.37
192.168.156.224
192.168.60.153
192.168.69.116
```

In this `with` statement, I used `"w"` as the second argument of the `open()` function. This argument indicates that I want to open the file to write over its contents. When this argument is used, the `.write()` function can be called within the body of the `with` statement. Calling this function allows you to overwrite the content of a file.

In this case, I wanted to overwrite the updated list of allowed IPs as a string in the `"allow_list.txt"` file. To overwrite the file, I added the `.write()` function to the `file` variable, which contains the text file. Then, I passed the `ip_addresses` variable as an

argument to specify that the contents of the file indicated in the `with` statement (`import_file`) should be replaced with the contents of the `ip_addresses` variable.

To effectively use this algorithm, it should be included as the body of a defined function that takes the file to be overwritten and the elements to be removed as parameters. This way, each time you want to use the algorithm, you only need to write the function's name and pass the arguments to it.

Summary

I have created a Python algorithm that removes the IP addresses listed in the variable `remove_list` from the `"allow_list.txt"` file of allowed IP addresses. To achieve this, the algorithm `opens` the `"allow_list.txt"` file, converting it into a string to read its contents, and then transforms it back into a list stored in the `ip_addresses` variable to allow iteration. This iteration is performed on the `ip_addresses` variable, with the condition that each iteration checks if the `element` is also in the `remove_list` variable. If it is, the `.remove()` method is applied to delete the `element` from `ip_addresses`. After this, the `.join()` method is used to convert the `ip_addresses` list back into a string. As a string, and using the `'w'` parameter, the contents of the `"allow_list.txt"` file can be overwritten with the revised content of the `ip_addresses` list.