Algorithm for file updates in Python

Project description

At my organization, access to restricted content is controlled with an allow list of IP addresses. The `"allow_list.txt"` file identifies these IP addresses. A separate remove list identifies IP addresses that should no longer have access to this content. I created an algorithm to automate update the `"allow_list.txt"` file and remove these IP addresses that should no longer have access.

Open the file that contains the allow list

I started by opening the `"allow_list.txt”` and assigned it to the `import_file` variable. Then, I used the with statement to open the file. The with is used with `.open()` for various purposes, in this case, reading a file. Below is the appropriate code with the correspondent descriptions:

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

import_file = "allow_list.txt"
```

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

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

Read the file contents

The next step was to use the `.read()` method to convert the file content into a string, so I can use it later on.

```
# Use `.read()` to read the imported file and store it in a variable named `ip_addresses`

ip_addresses = file.read()
```

Convert the string into a list

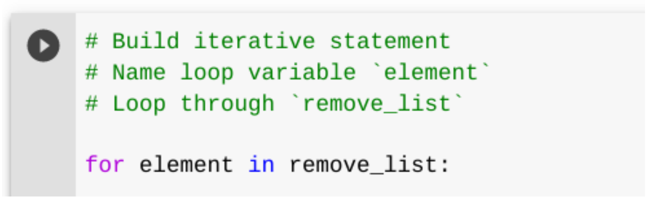
I simply used the `.split()` split method, which automatically converts the contents of a string into a list. Each element of the list comes from a white space in the string.

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

ip_addresses = ip_addresses.split()
```

Iterate through the remove list

In order to make this step possible, I needed to implement a `for` loop that iterates through the IP addresses that are elements of the `remove_list`. The `for` loop in Python repeats code for a specified sequence.



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

for element in remove_list:
```

Remove IP addresses that are on the remove list

In this step I filtered the list to remove the flagged addresses from the `remove_list`. I had to use an if statement to search if each IP address was in the `remove_list`, and if true, then use the method `.remove()` to do this.

```python
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)
```

Update the file with the revised list of IP addresses

I finished this algorithm by updating the original list after removing the flagged IP addresses. First I had to convert the list back into a string by using the `.join()` method.

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

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

Lastly, I used another with statement and the write method to update the file. This step is the exact opposite to the initial.

```python
# 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)
```

Summary

And this is how I implemented this relatively simple but efficient program which can be used to update the IP addresses allowed to enter restrictive data. In brief, this algorithm involves opening and reading a file, convert the contents into a list, remove the flagged addresses from the list, and finally update the original list so it doesn’t allow potentially dangerous IP addresses to access sensitive data.

*Final Note: Feel free to check out the Reference Python Notebook shared along with this project in case you would like to see all the steps followed in further detail.