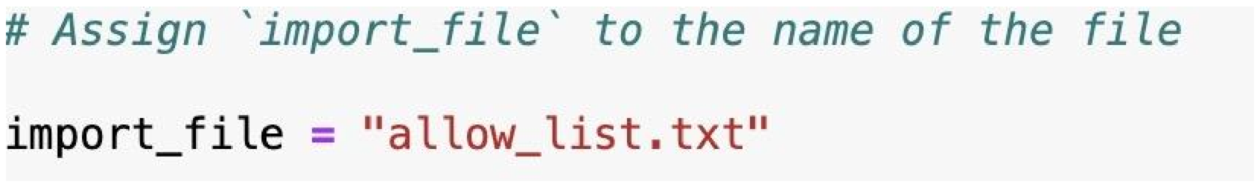
Algorithm for file updates in Python

Project description

At my workplace, access to sensitive resources is managed through an allow list of IP addresses stored in the `"allow_list.txt"` file. There is also a separate list that specifies IP addresses which should have their access revoked. I developed a script that automates the process of updating the allow list by removing any IP addresses that are no longer permitted.

Open the file that contains the allow list

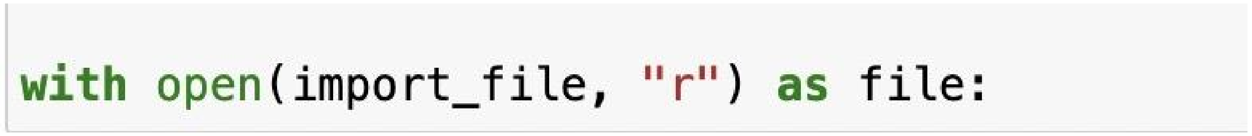
To create the script/algorithm first I created a variable and stored the name of the file `"allow_list.txt"` in it. I named the variable `import_file`



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

import_file = "allow_list.txt"
```

Then i used with statement to open the file



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

In my algorithm, I use a `with` statement along with the `open()` function in **read mode** to access the allow list file. Opening the file in this way lets me retrieve the IP addresses stored inside it. The `with` keyword is useful because it automatically handles resource management by closing the file once the block of code is finished. For example, in the line `with open(import_file, "r") as file:`, the `open()` function takes two arguments: the first specifies the file to be opened, and the second defines the mode—in this case, `”r”` means the file is opened for reading. The `as` keyword assigns the file object to the variable `file`, which I then use to interact with the contents while inside the `with` block.

Read the file contents

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

```python
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 the `open()` function is used with the `"r"` mode, the `.read()` method can be called inside the `with` block to capture the file's contents. This method takes everything in the file and returns it as a single string. In my code, I applied `.read()` to the `file` object defined in the `with` statement, and then stored the resulting string in a variable called `ip_addresses`.

In short, this process loads all of the data from `"allow_list.txt"` into a string, which I can later manipulate in my Python program to filter, structure, or extract the information I need.

Convert the string into a list

Next I used the `.split()` method to convert the `ip_addresses` list into a string, because I need it to be in list format to be able to remove individual IP addresses from the allow list.

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

ip_addresses = ip_addresses.split()
```

I used the `.split()` method to turn the string of IP addresses into a list, making it easier to manage and remove specific entries. Since the IPs in the file were separated by spaces, `.split()` automatically broke them into individual list elements, which I reassigned back to `ip_addresses` for further processing.

Iterate through the remove list

An important step in my algorithm is looping through the IP addresses stored in `remove_list`. To handle this, I used a `for` loop.

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

for element in ip_addresses:
```

I used a `for` loop to go through each IP address in the list, assigning them one by one to the variable `element` so the algorithm could check and process each value.

Remove IP addresses that are on the remove list

My algorithm needed to filter out any IP addresses in `ip_addresses` that also appeared in `remove_list`. Since there were no duplicates in `ip_addresses`, I could achieve this with the following code:

```
for element in remove_list:

  # Build conditional statement
  # If current element is in `remove_list`,

    if element in ip_addresses:

        # then current element should be removed from `ip_addresses`

        ip_addresses.remove(element)
```

Inside the loop, I added a condition to check if the current `element` existed in the `ip_addresses` list. This safeguard was necessary because trying to remove a value that isn't present would cause an error. When the condition was true, I used the `.remove()` method with `element` as the argument, ensuring that any IP address found in `remove_list` was properly taken out of `ip_addresses`.

Update the file with the revised list of IP addresses

The last step in my algorithm was to refresh the allow list file with the updated IP addresses. Since the data was stored as a list, I first converted it back into 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)
```

The `.join()` function is used to merge the items of a list into a single string, with a chosen separator placed between each item. In my algorithm, I applied `.join()` to the `ip_addresses` list so it could be written back into the file. By using `"\n"` as the separator, I ensured that every IP address would appear on its own line in the updated `"allow_list.txt"` file.

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

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

  # Rewrite the file, replacing its contents with `ip_addresses`

  file.write(ip_addresses)
```

For this step, I opened the file in write mode by passing `"w"` as the second argument to the `open()` function inside the `with` statement. Using `"w"` allows me to overwrite the file's existing contents. Inside the block, I called the `.write()` method on the `file` object to save the updated data. By passing in the `ip_addresses` string, the `"allow_list.txt"` file was rewritten with the revised list of IPs, ensuring that any addresses removed from the allow list no longer had access to restricted resources.

Summary

I developed an algorithm to update the `"allow_list.txt"` file by removing any IP addresses listed in the `remove_list` variable. The process began by opening the file and reading its contents into a string, which I then split into a list called `ip_addresses`. Next, I looped through each item in `remove_list` and checked if it appeared in `ip_addresses`. When a match was found, the `.remove()` method was applied to delete that entry. Finally, I converted the updated list back into a string with `.join()` and overwrote the original file, ensuring the allowed list only contained valid IP addresses.