Algorithm for file updates in Python

Project description

As a security analyst in this organization, there is a file called `allow_list.txt` that contains allowed IP addresses. I must set up an algorithm to remove some IP addresses, this is completed through the `remove_list` file. Those who are in remove_list will no longer have access while those in the `allow_list` continue to have access.

Open the file that contains the allow list

For the first part of the algorithm, I opened the `"allow_list.txt"` file. First, I assigned this file name as a string to the `import_file` variable:

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

import_file = "allow_list.txt"
```

Then, 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:
```

In my algorithm, I use the `with` statement together with the `open()` function in read mode (`"r"`) to open the allow list file. This lets me access the IP addresses stored inside that file. The `with` keyword is helpful because it automatically handles resource management—specifically, it closes the file as soon as the `with` block is exited.

When I write `with open(import_file, "r") as file:`, the `open()` function is given two arguments: the first specifies the file I want to work with, and the second tells Python what action to perform with it. In this case, `"r"` means the file should be opened for reading. The `as` keyword assigns the opened file object to a variable called `file`, which I can then use to work with the file's contents while inside the `with` block.

Read the file contents

I had to use the `.read()` method to convert it into the string in order to read the file contents.

```
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()
```

Using `open()` in read mode argument (`"r"`), I called `.read()` inside the `with` block to load the entire file as a string into `ip_addresses`. This allows me to later organize and extract data from `"allow_list.txt"` in my Python program. This code reads the contents of the "allow_list.text" file into a string format for later use of the string to extract and read the data.

Convert the string into a list

I can reorder to remove individual IP addresses from the allow list, I needed it to be in list format. Therefore, I next used the `.split()` method to convert the `ip_addresses` string into a list:

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

ip_addresses = ip_addresses.split()
```

The `.split()` method is invoked by appending it to a string variable, where it transforms the string's contents into a list. In this case, splitting `ip_addresses` into a list makes it easier to remove specific IP addresses from the allow list. By default, `.split()` separates text at whitespace, creating individual list elements. In this algorithm, `.split()` operates on the `ip_addresses` string—which contains multiple IP addresses separated by spaces—and converts it into a list of individual addresses. This new list is then reassigned to the `ip_addresses` variable for further processing.

Iterate through the remove list

My algorithm involves iterating through the IP addresses that are elements in the `remove_list`. To do this, I incorporated a `for` loop:

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

for element in remove_list:
```

In Python, the `for` loop repeats code for a specified sequence. In an algorithm like this, its purpose is to apply specific code statements to every element in the sequence. The `for` keyword begins the loop, followed by the loop variable `element` and the keyword `in`. The keyword `in` instructs Python to iterate through the sequence `ip_addresses`, assigning each value to the loop variable `element`.

Remove IP addresses that are on the remove list

Since there were no duplicate entries in `ip_addresses`, I was able to remove any addresses that also appeared in `remove_list` using the following code. My algorithm works by checking `ip_addresses` against `remove_list` and removing any matches from the allow list.

```
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)
```

Inside the `for` loop, I first added a condition to check whether the loop variable `element` existed in the `ip_addresses` list. This safeguard prevents errors that would occur if `.remove()` were called on an item not present in the list.

If the condition was met, I used the `.remove()` method on `ip_addresses`, passing `element` as the argument. This ensured that each IP address listed in `remove_list` was successfully deleted from `ip_addresses`.

Update the file with the revised list of IP addresses

In the final step of my algorithm, I updated the allow list file with the revised list of IP addresses. To do so, I first needed to convert the list back into a string. I used the `.join()` method for this:

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

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

I used the `.join()` method to transform the `ip_addresses` list into a single string, preparing it for writing back to `"allow_list.txt"`. The `.join()` method works by combining all items in an iterable, using the string it's called on as the separator. In my case, I used `"\n"` as the separator so that each IP address would appear on its own line in the file. This approach ensured that the output was neatly formatted and ready for use.

Additionally, I had 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)
```

In this step, I opened `"allow_list.txt"` in write mode by passing `"w"` as the second argument to the `open()` function within my `with` statement. Using `"w"` tells Python to overwrite the file's contents. Inside the `with` block, I used the `.write()` method to store new data in the file. Since `.write()` replaces everything in the file, I passed in the `ip_addresses` variable, which contained the updated allow list formatted as a string. This ensured that any IP addresses removed from the list could no longer access the restricted content.

Summary

I developed an algorithm to update the approved IP addresses in `"allow_list.txt"` by removing entries found in a `remove_list` variable. First, I opened the file and read its contents as a string, then split it into a list assigned to `ip_addresses`. I iterated through each address in `remove_list`, checking if it existed in `ip_addresses`. If found, I removed it using `.remove()`. Once all removals were complete, I used `.join()` to convert the list back into a string and overwrote `"allow_list.txt"` with this updated content, ensuring removed IPs no longer had access.