Algorithm for file updates in Python

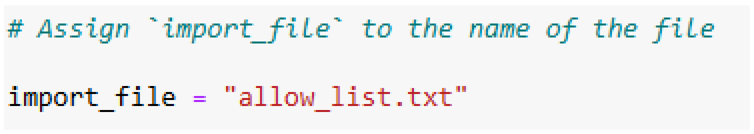
By Jack McGrath

Project description

At my organization, access to restricted content is determined by a list of IP addresses. The file `“allow_list.txt”` identifies these IP addresses, as well as a separate remove list identifies who should not have access. I created an algorithm that automated the updating required for the allow list, to easily remove access.

Open the file that contains the allow list

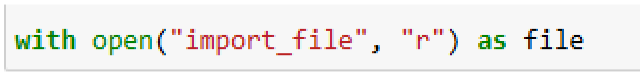
The first part of the algorithm involved opening the file that contains the `“allow_list.txt”`. I assigned the file name as a string to the `import_file` variable as seen below:



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

import_file = "allow_list.txt"
```

Then using the with statement, I opened the file.

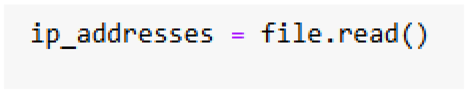


```
with open("import_file", "r") as file
```

In my algorithm, the `with` statement is used in combination with the `.open()` function in read mode to access the allow list file. This enables me to retrieve the IP addresses stored within it. The code `with open(import_file, "r") as file:` includes two parameters in the `open()` function: the first specifies the file to be accessed, and the second, `"r"`, indicates that the file should be opened in read mode. The `as` keyword assigns the file object to the variable `file`, allowing me to interact with the file’s contents while inside the `with` block.

Read the file contents

In order to read the file contents, I used `.read()` to convert the contents of the file into a string.



```
ip_addresses = file.read()
```

When using the `.open()` function with the `"r"` argument for reading, I can call the `.read()` method within the `with` statement. The `.read()` method converts the file's contents into a string, making it possible to read and process the data. The output of `file.read()` is assigned to ip_addresses. This means that when I call ip_addresses, what is being returned is the output of `file.read()`, allowing me to read the contents of `“allow_list.txt”`

Convert the string into a list

In order to have the algorithm be able to remove IP addresses, the string had to become a list. This was achieved by using the `.split()` function.

```
ip_addresses = ip_addresses.split()
```

`.split()` converts the contents of a string into a list by splitting the string at each whitespace. This makes it far easier to read and also to identify what IP addresses had to be removed.

Iterate through the remove list

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

```
for element in remove_list:
```

A for loop works by executing a block of code for each item specified in a sequence.
The loop begins with the `for` keyword, followed by a loop variable (`element`) and the `in` keyword. The `in` keyword signals iteration through the `remove_list` sequence, assigning each value to the loop variable `element` in each iteration.

Remove IP addresses that are on the remove list

My algorithm needs to remove any IP address from the `ip_addresses` allow list that also appears in `remove_list`. Since `ip_addresses` initially contained no duplicates, I was able to accomplish this using 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)

# Display `ip_addresses`

print(ip_addresses)
```

Within my `for` loop, I first included a conditional statement to check if the loop variable `element` existed in the `ip_addresses` list. This step was necessary to prevent errors that would occur if `.remove()` were called on a non-existent element.

If the condition was met, I then applied the `.remove()` method to `ip_addresses`, using `element` as the argument. This ensured that any IP address found in `remove_list` was successfully removed from `ip_addresses`.

Update the file with the revised list of IP addresses

As the final step in my algorithm, I had to update the allow list file with the updated list of IP addresses. To achieve this, I first needed to convert the list back into a string. I used the `.join()` method to accomplish this:

```
# 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 single string. It is applied to a string that specifies how the elements will be separated when joined.

I used the `.join()` method to turn the `ip_addresses` list into a string so that it could be passed as an argument to the `.write()` method when updating the `"allow_list.txt"` file. I used the separator `("\n")` to ensure that each IP address would be placed on a separate line.

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)
```

In this step, I used the `"w"` argument with the `open()` function inside the `with` statement. This argument tells Python to open the file for writing, allowing me to overwrite its existing contents. With this mode, I could call the `.write()` method inside the `with` block. The `.write()` method writes string data to the file and replaces anything that was already there.

For this algorithm, I needed to write the updated allow list as a string to `”allow_list.txt”` ensuring that any IP addresses removed from the list would no longer have access. To update the file, I used the `.write()` method on the `file` object I defined in the `with` statement, passing in the `ip_addresses` variable to replace the old content with the new data.

Summary

I developed an algorithm that removes IP addresses listed in a `remove_list` variable from the `”allow_list.txt”` file, which contains approved IP addresses. The algorithm first opens the file, converts its contents into a string, and then turns that string into a list stored in the `ip_addresses` variable. Next, I looped through the IP addresses in the `remove_list` and checked if each element was present in the `ip_addresses` list. If it was, I used the `.remove()` method to remove it. Afterward, I applied the `.join()` method to convert the `ip_addresses` list back into a string, allowing me to overwrite the contents of `”allow_list.txt”` with the updated list of IP addresses.