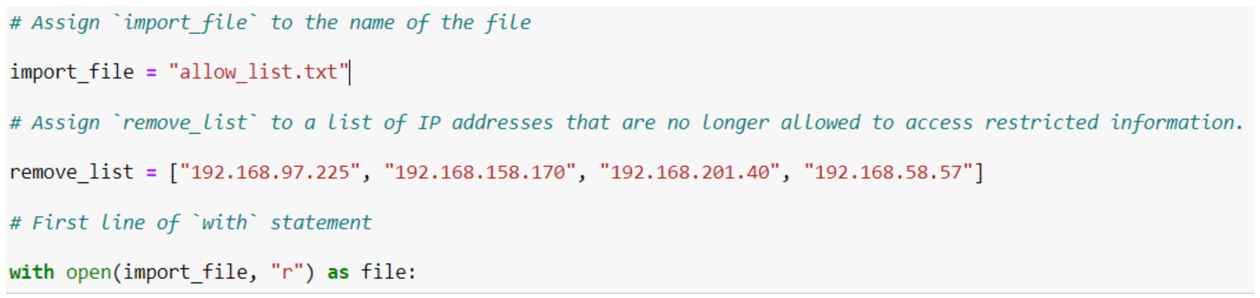
Algorithm for file updates in Python

Project description

At my organization, we manage access to restricted content using an IP address whitelist, which is maintained in a file named `allow_list.txt.` This file contains the IP addresses that are authorized to access the content. Additionally, we maintain a separate list, known as the `remove_list,` which identifies IP addresses that should no longer have access to this content. To streamline the process of managing access, I've developed an algorithm that automates the removal of these outdated IP addresses from the `allow_list.txt` file.

Open the file that contains the allow list

In order to open the file that contains the allow list, we put the name of the file into the variable `import_file`, then we use the `with` statement to open the file containing the file `allow_list.txt` .

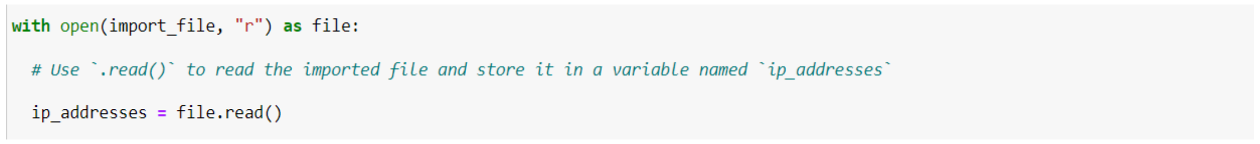


```python
# Assign `import_file` to the name of the file

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"]

# First line of `with` statement

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

In my algorithm, I utilize the `with` statement in conjunction with the `.open()` function set to read mode to access and open the allow list file. The primary objective of this action is to gain access to the IP addresses stored within the allow list file. The utilization of the `with` keyword in this context also serves to efficiently manage resources, as it automatically closes the file upon exiting the `with` statement.

Read the file contents

In order to read the contents of the file, we use the `.read()` method.



```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 utilizing an `.open()` function with the `r` argument for `read,` I can employ the `.read()` function within the context of the `with` statement. The purpose of the `.read()` method is to transform the file's contents into a string, facilitating its readability. In this specific scenario, I applied the `.read()` method to the file variable specified within the `with` statement. Subsequently, I captured the resulting string output by assigning it to the variable named `ip_addresses.`

Convert the string into a list

We must use the `.split()` method to convert the string into a list. This makes it possible to change the list since strings are immutable. To use this, we take the variable `ip_addresses` and add the method to the end of it, storing this in a variable aptly called `ip_addresses.` We then display the variable using the `print()` function.

```
# 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 iterate through the remove list, the `for` loop is used. We name the loop variable `element` and loop through `ip_addresses.`

```
# 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"]

# 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()

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

ip_addresses = ip_addresses.split()

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

for element in ip_addresses:

    # Display `element` in every iteration

    print(element)
```

In Python, a `for` loop iterates over a defined sequence, executing a set of code statements for each element within that sequence. In the context of a Python algorithm like this one, the primary objective of the `for` loop is to systematically apply designated code instructions to every element present in a sequence.

The `for` loop commences with the `for` keyword, followed by the loop variable (typically named `element` for clarity), and the `in` keyword. The `in` keyword signals the loop to iterate through the specified sequence, in this case, `ip_addresses,` and assigns each element's value to the loop variable, which is denoted as `element.`

Remove IP addresses that are on the remove list

Removing IP addresses that are on the remove list requires the `.remove()` method. Using this method and inputting specific parameters in the parenthesis allows us to remove aspects of lists asfwithin said parameters.

```
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)
```

In my `for` loop, I initiated a conditional check to assess whether the loop variable `element` exists within the `ip_addresses` list. This step was necessary to prevent potential errors that could arise from attempting to apply the `.remove()` method to elements not present in the `ip_addresses` list.

Within this conditional statement, I proceeded to execute the `.remove()` method on the `ip_addresses` list. To ensure the removal of specific IP addresses identified in the `remove_list,` I used the loop variable `element` as the argument for the `.remove()` method. This way, each IP address matching an entry in the remove_list was effectively eliminated from the `ip_addresses` list.

Update the file with the revised list of IP addresses

The `.join()` method serves the purpose of merging all items within an iterable into a single string. To use the `.join()` method effectively, it is applied to a string containing characters that dictate how the elements within the iterable should be separated once they are amalgamated into a string.

In this particular algorithm, I employed the `.join()` method to create a string from the `ip_addresses` list. This string was then utilized as an argument for the `.write()` method when updating the allow_list.txt file. To establish a clear separation between elements in the final string, I employed the newline character " " as the separator.

I employed another with statement along with the .write() method to carry out the file update. This time, I utilized a second argument of "w" within the open() function of the with statement. This "w" argument signifies the intent to open the file for writing, effectively replacing its existing contents.

The .write() function is responsible for writing string data to a specified file while overwriting any pre-existing content within that file. In this context, my goal was to write the updated allow list as a string to the allow_list.txt file. To achieve this, I appended the .write() function to the file object, which I had previously identified in the with statement. By passing the ip_addresses variable as the argument, I specified that the contents of the file, as designated in the with statement, should be entirely replaced with the data contained in this variable. This process ensures that the restricted content becomes inaccessible to any IP addresses that were removed from the allow list, effectively rewriting the file with the updated information.

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

ip_addresses = " ".join(ip_addresses)

# Build `with` statement to rewrite the original file

with open(import_file, "w"):

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

  file.write(ip_addresses)
```

Summary

I designed an algorithm to extract specified IP addresses from the allow_list.txt file, which contains approved IP addresses. This involved a series of steps, including file opening, conversion of its content into a string for reading, and subsequent transformation of this string into a list stored as ip_addresses.

I then systematically checked each IP address in the remove_list. During each iteration, I verified if the element existed in the ip_addresses list. If it did, I utilized the .remove() method to eliminate the element from ip_addresses.

Following this removal process, I employed the .join() method to convert ip_addresses back into a string format. This string was then used to overwrite the contents of the allow_list.txt file, updating it with the revised list of IP addresses.

```python
# Define a function named `update_file` that takes in two parameters: `import_file` and `remove_list`
# and combines the steps you've written in this lab leading up to this

def update_file(import_file, remove_list):

  # 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()

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

  ip_addresses = ip_addresses.split()

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

  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)

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

  ip_addresses = " ".join(ip_addresses)

  # 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)

# Call `update_file()` and pass in "allow_list.txt" and a list of IP addresses to be removed

update_file("allow_list.txt", ["192.168.25.60", "192.168.140.81", "192.168.203.198"])

# Build `with` statement to read in the updated file

with open("allow_list.txt", "r") as file:

  # Read in the updated file and store the contents in `text`

  text = file.read()

# Display the contents of `text`

print(text)
```