Algorithm for file updates in Python

Project description

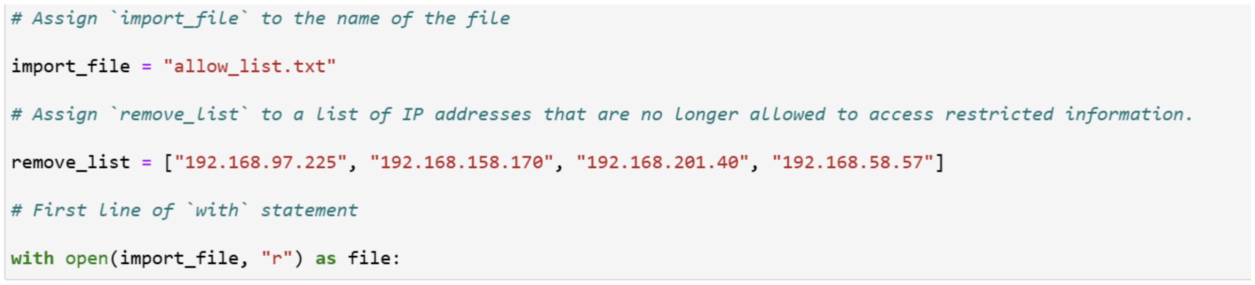
Overview:
At my organization, access to restricted content is governed by an allow list of approved IP addresses, maintained in a file named `allow_list.txt`. To ensure security and streamline updates, I developed an algorithm that automatically removes IP addresses listed in a separate `remove_list`, representing users who should no longer have access.

Key Features:

- **File Handling:** Reads the current allow list from `allow_list.txt` using Python's `with open()` statement in read mode.
- **Data Transformation:** Converts the file content into a list of IP addresses for easy manipulation.
- **Filtering Logic:** Iterates through the `remove_list` and conditionally removes matching IPs from the allow list using the `.remove()` method.
- **File Update:** Converts the updated list back into a string using `.join()` and overwrites the original file using write mode to reflect the changes.

Impact:
This solution automates a previously manual process, reduces human error, and ensures that unauthorized IP addresses are promptly removed from the system, enhancing overall data security and operational efficiency.

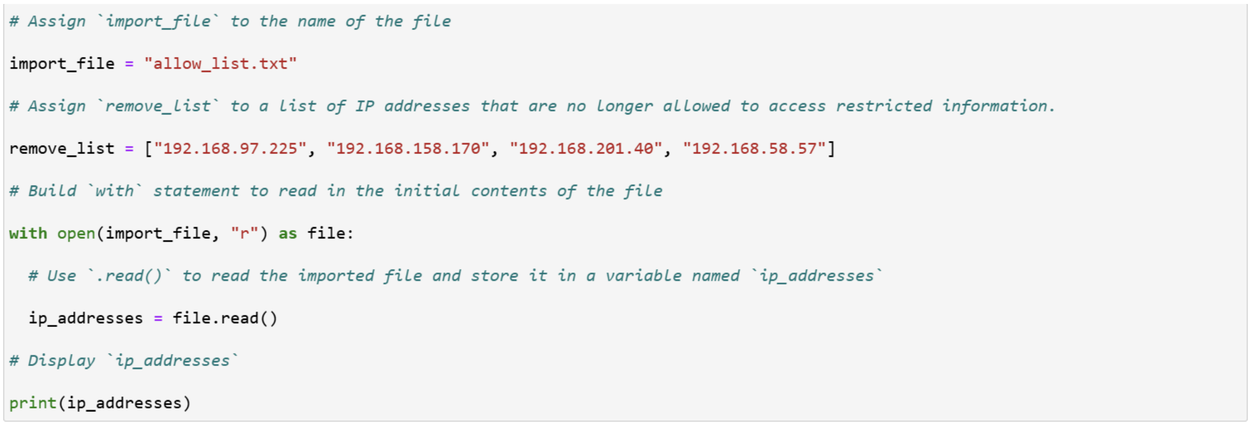
Open the file that contains the allow list



```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 use the `with` statement in combination with the `.open()` function in read mode to access the allow list file. This approach enables me to retrieve the IP addresses stored within the file. The `with` keyword ensures proper resource management by automatically closing the file once the block is exited. The line `with open(import_file, "r") as file:` includes two arguments in the `open()` function: the first specifies the file to be opened, and the second—`"r"`—indicates that the file should be read. The `as` keyword assigns the opened file object to the variable `file`, which I then use to interact with the file's contents during the execution of the `with` block

Read the file contents

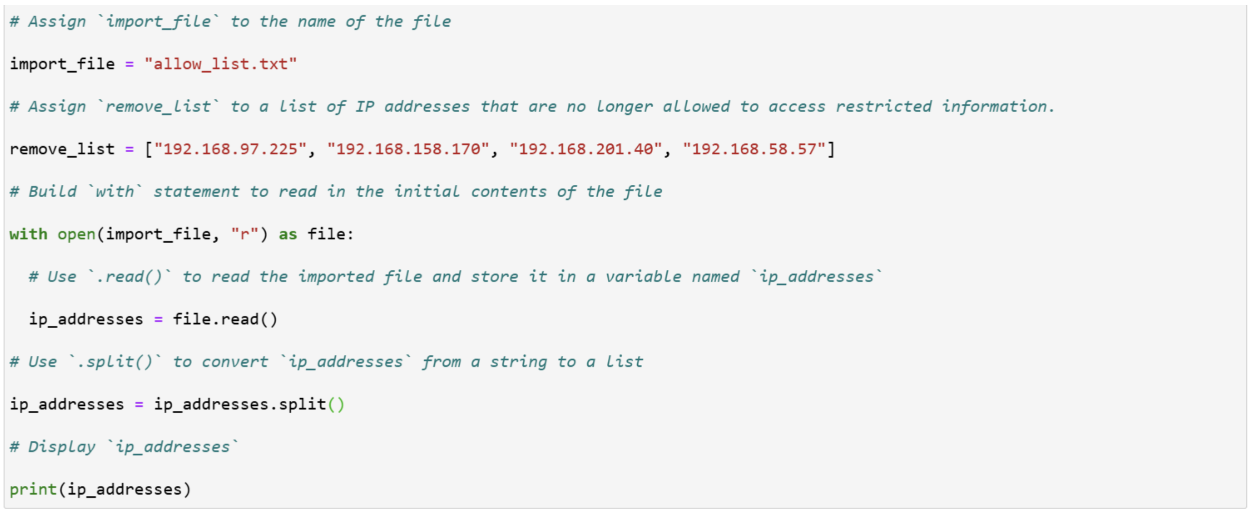


```python
# Assign `import_file` to the name of the file

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"]

# 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()

# Display `ip_addresses`

print(ip_addresses)
```

When I use the `open()` function with the `"r"` argument for reading, I can call the `.read()` method inside the `with` block to get the file's contents as a string. I apply `read()` to the `file` variable from the `with` statement and store the result in a new variable called `ip_addresses`.

In short, this code reads everything from the `"allow_list.txt"` file into a string, which I can later use to organize and extract data in my Python program.

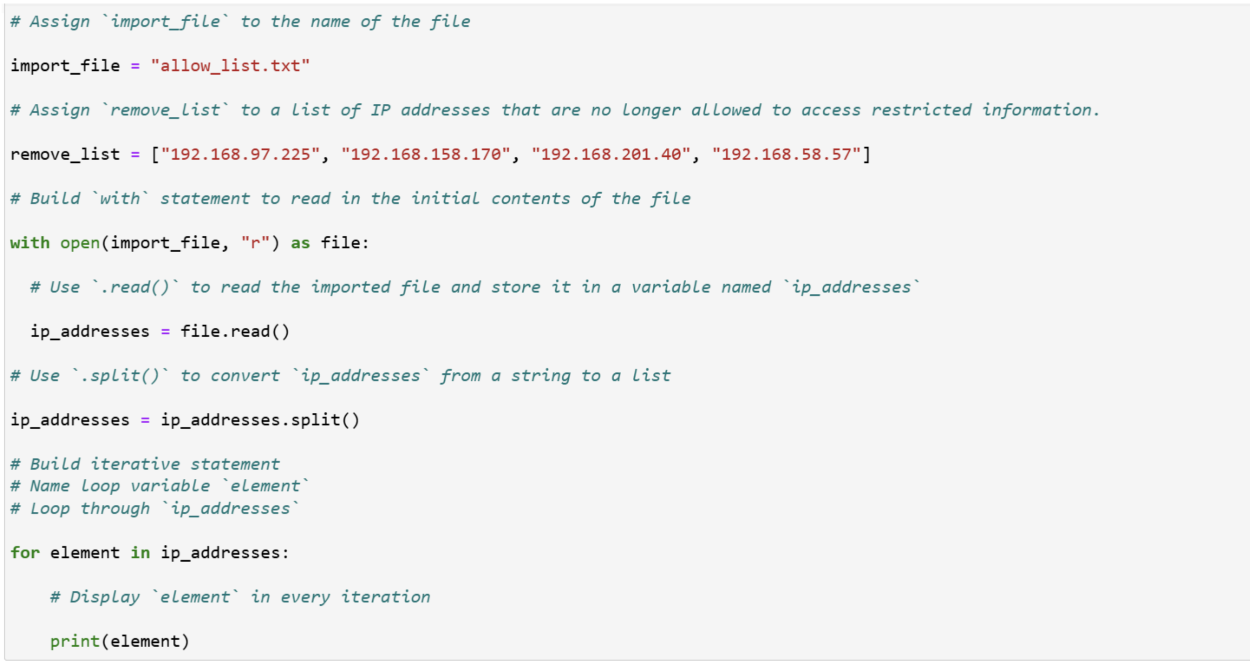
Convert the string into a list



```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"]

# 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()

# Display `ip_addresses`

print(ip_addresses)
```

The `.split()` method is used on a string to break it into a list. In this case, I applied it to the `ip_addresses` string, which contains IPs separated by spaces. By default, `.split()` uses whitespace to divide the string into individual elements. This makes it easier to manage and remove IP addresses from the allow list. After splitting, I stored the resulting list back into the `ip_addresses` variable.

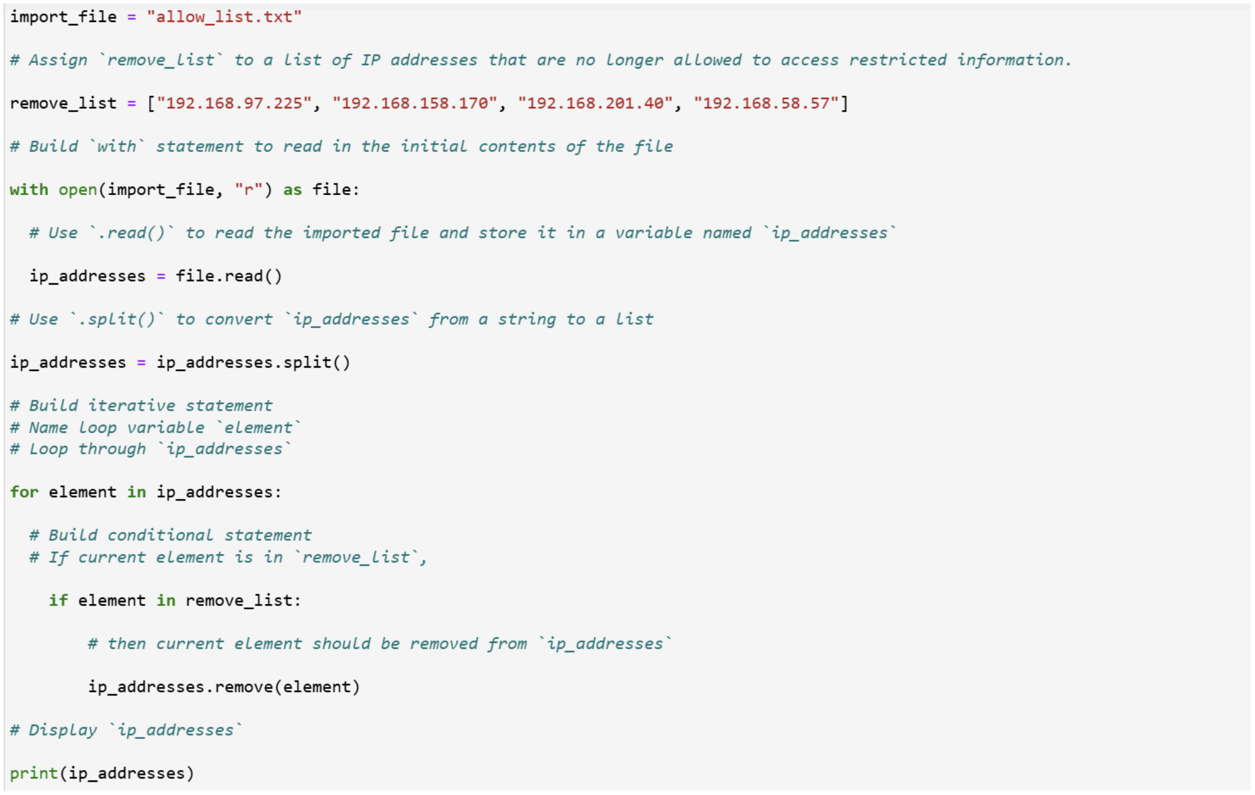
Iterate through the remove list



```
# 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)
```

A key part of my algorithm is looping through the IP addresses listed in `remove_list`. I use a `for` loop to do this. In Python, a `for` loop runs a block of code for each item in a sequence. The loop starts with the `for` keyword, followed by a variable name (like `element`), and the keyword `in`, which tells Python to go through each item in the sequence—such as `ip_addresses`—and assign it to the loop variable one at a time.

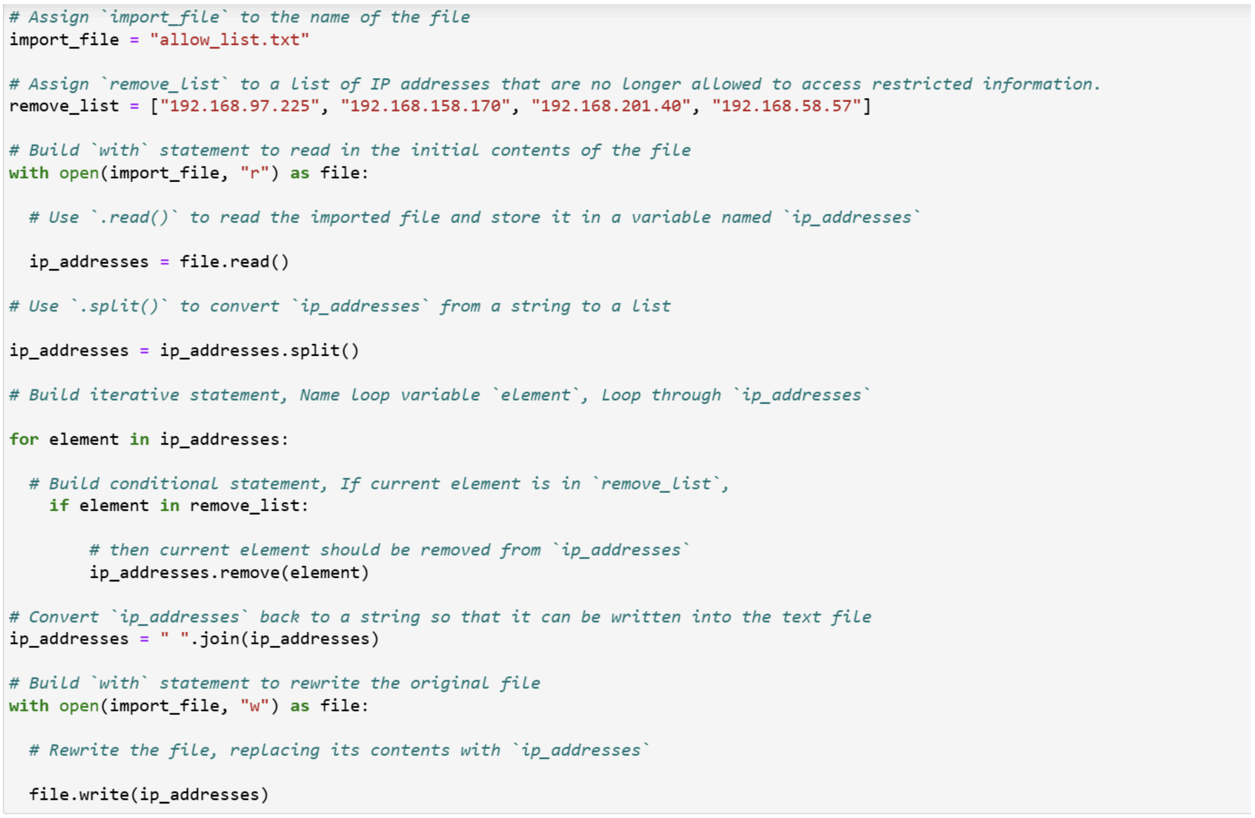
Remove IP addresses that are on the remove list



```python
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:

  # 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)

# Display `ip_addresses`

print(ip_addresses)
```

My algorithm removes any IP address from the `ip_addresses` list if it also appears in the `remove_list`. Since `ip_addresses` had no duplicates, I used a `for` loop to check each item in `remove_list`. Inside the loop, I added a condition to see if the current IP (stored in the loop variable) exists in `ip_addresses`. This prevents errors from trying to remove items that aren't there. If the condition is true, I call `.remove()` on `ip_addresses`, passing in the current IP to delete it from the list.

Update the file with the revised list of 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:

  # 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)
```

To finish my algorithm, I needed to update the `"allow_list.txt"` file with the revised IP addresses. First, I converted the `ip_addresses` list back into a string using the `.join()` method. This method combines all items in a list into a single string, using a specified separator—in this case, `"\n"`—to place each IP address on a new line.

Next, I opened the file in write mode using `open("allow_list.txt", "w")` inside a `with` statement. The `"w"` argument tells Python to overwrite the file's contents. Inside the `with` block, I used the `.write()` method on the file object to save the updated string. By passing in the `ip_addresses` string, the file was rewritten with the new list, ensuring that any removed IPs no longer have access.

Summary

I developed an algorithm to remove IP addresses listed in the `remove_list` variable from the `"allow_list.txt"` file, which contains approved IPs. The process begins by opening the file and reading its contents as a string. This string is then split into a list and stored in the `ip_addresses` variable. I loop through each IP in `remove_list`, checking if it exists in `ip_addresses`. If it does, I use the `.remove()` method to delete it. Once all necessary IPs are removed, I convert the updated list back into a string using the `.join()` method. Finally, I overwrite the original file with this revised string, ensuring the removed IPs no longer have access.