Algorithm For File Updates In Python

Project description

I work as a security professional at a healthcare company, and one of my regular tasks involves managing a file that contains a list of employees authorized to access sensitive patient records. Access to this confidential information is controlled by filtering employees' IP addresses.

I have two important lists at my disposal:

1. **Allow List**: This contains the IP addresses of employees who are permitted to access restricted patient records.

2. **Remove List**: This list specifies which employees should no longer have access to these records.

My job is to develop a Python algorithm to automatically check if any of the IP addresses in the "Remove List" are present in the "Allow List." If there is a match, I must remove those IP addresses from the "Allow List" file. This automation ensures that only authorized employees have access to sensitive patient data, maintaining the security and privacy of our healthcare information.

Open the file that contains the allow list

Here's a breakdown of the syntax and keywords:

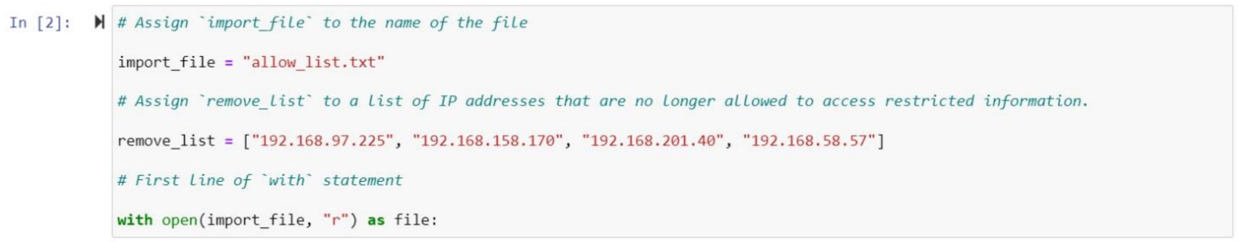
1. **import_file** is assigned the string **"allow_list.txt"** to specify the name of the file you want to open.

2. **with open(import_file, "r") as file:**:

   - **with** is a context manager in Python. It ensures that resources, in this case, the file, are properly managed and closed when you're done with them.

   - **open(import_file, "r")** opens the file specified by **import_file** in read-only mode ("r").

   - **as file** assigns the opened file to the variable **file** within the context of the **with** statement.



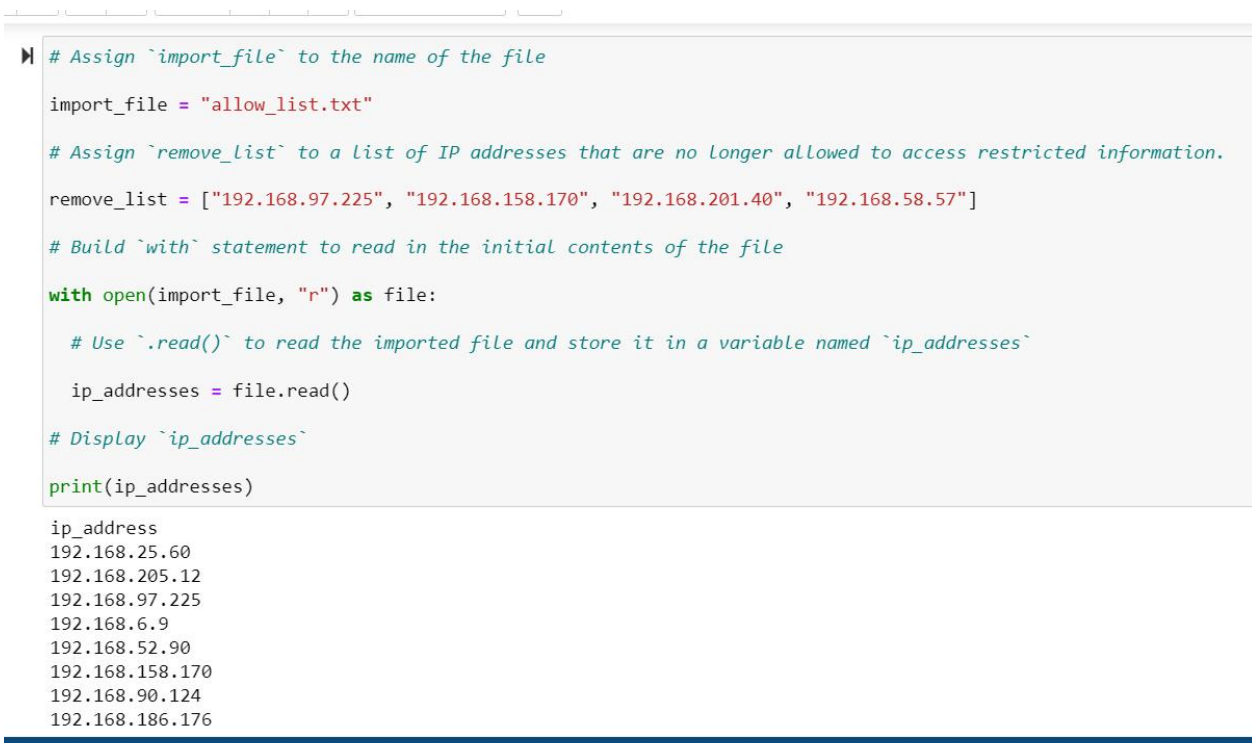
```
In [2]:  # Assign `import_file` to the name of the file

         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:
```

Read the file contents

Here's a breakdown of the syntax and keywords:

1. **ip_addresses** is a variable where you store the content of the file.
2. **file** is the variable representing the opened file that you wish to read.
3. **.read()** is a method used to read the entire contents of the file.

So, to read the contents of the file into the **ip_addresses** variable, you just call **.read()** on the **file** object and assign the result to **ip_addresses**.

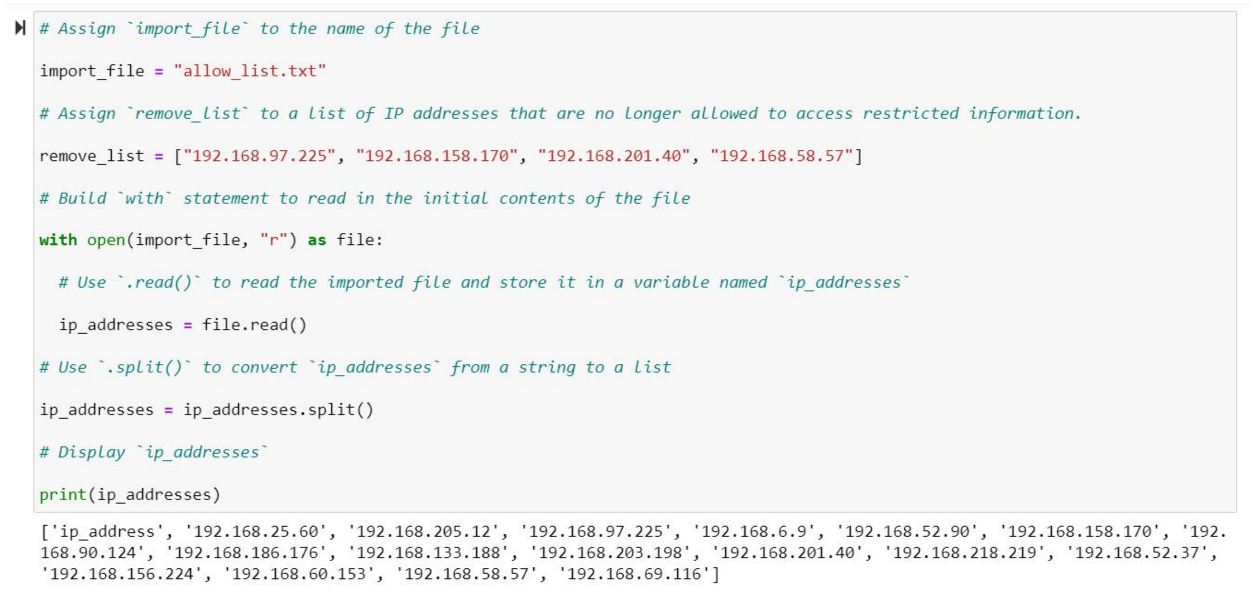


```python
# Assign `import_file` to the name of the file

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)
```

```
ip_address
192.168.25.60
192.168.205.12
192.168.97.225
192.168.6.9
192.168.52.90
192.168.158.170
192.168.90.124
192.168.186.176
```

Convert the string into a list

This code takes the string in the **ip_addresses** variable and splits it into a list of individual IP addresses based on spaces. It updates the **ip_addresses** variable to store this list of IP addresses.

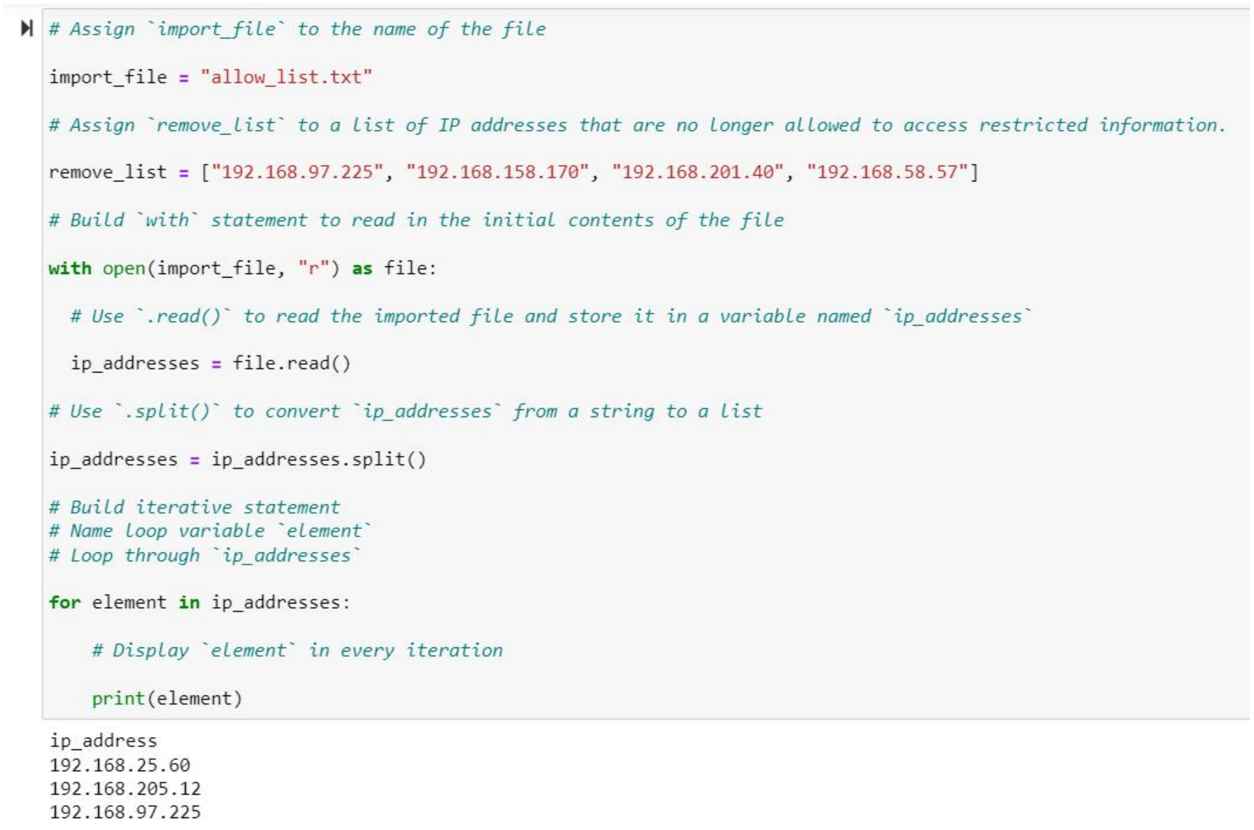


```
# Assign `import_file` to the name of the file

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()

# Display `ip_addresses`

print(ip_addresses)
```

```
['ip_address', '192.168.25.60', '192.168.205.12', '192.168.97.225', '192.168.6.9', '192.168.52.90', '192.168.158.170', '192.
168.90.124', '192.168.186.176', '192.168.133.188', '192.168.203.198', '192.168.201.40', '192.168.218.219', '192.168.52.37',
'192.168.156.224', '192.168.60.153', '192.168.58.57', '192.168.69.116']
```

Iterate through the remove list

In this code, the **for** loop iterates through each element in the **remove_list**, and during each iteration, you can perform actions or checks related to that specific element.



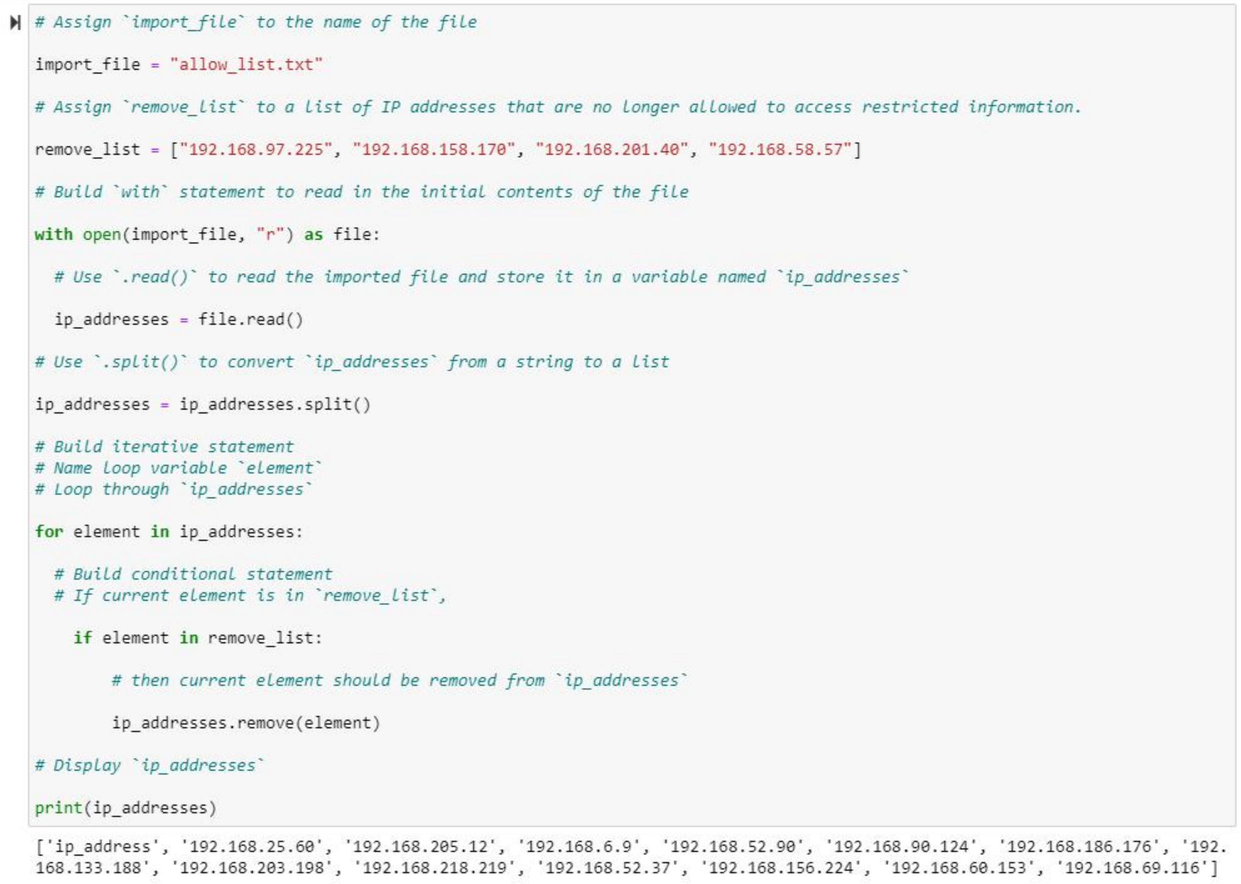
```python
# Assign `import_file` to the name of the file

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()

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

for element in ip_addresses:

    # Display `element` in every iteration

    print(element)
```

```
ip_address
192.168.25.60
192.168.205.12
192.168.97.225
```

Remove IP addresses that are on the remove list

This code checks if the **element** (IP address) from the **remove_list** exists in the **ip_addresses** list, and if it does, it removes that element from the **ip_addresses** list.

The reason you can use the **.remove()** method in this way is that there are no duplicates in the **ip_addresses** list. If there were duplicates, this method would remove the first occurrence of the element.

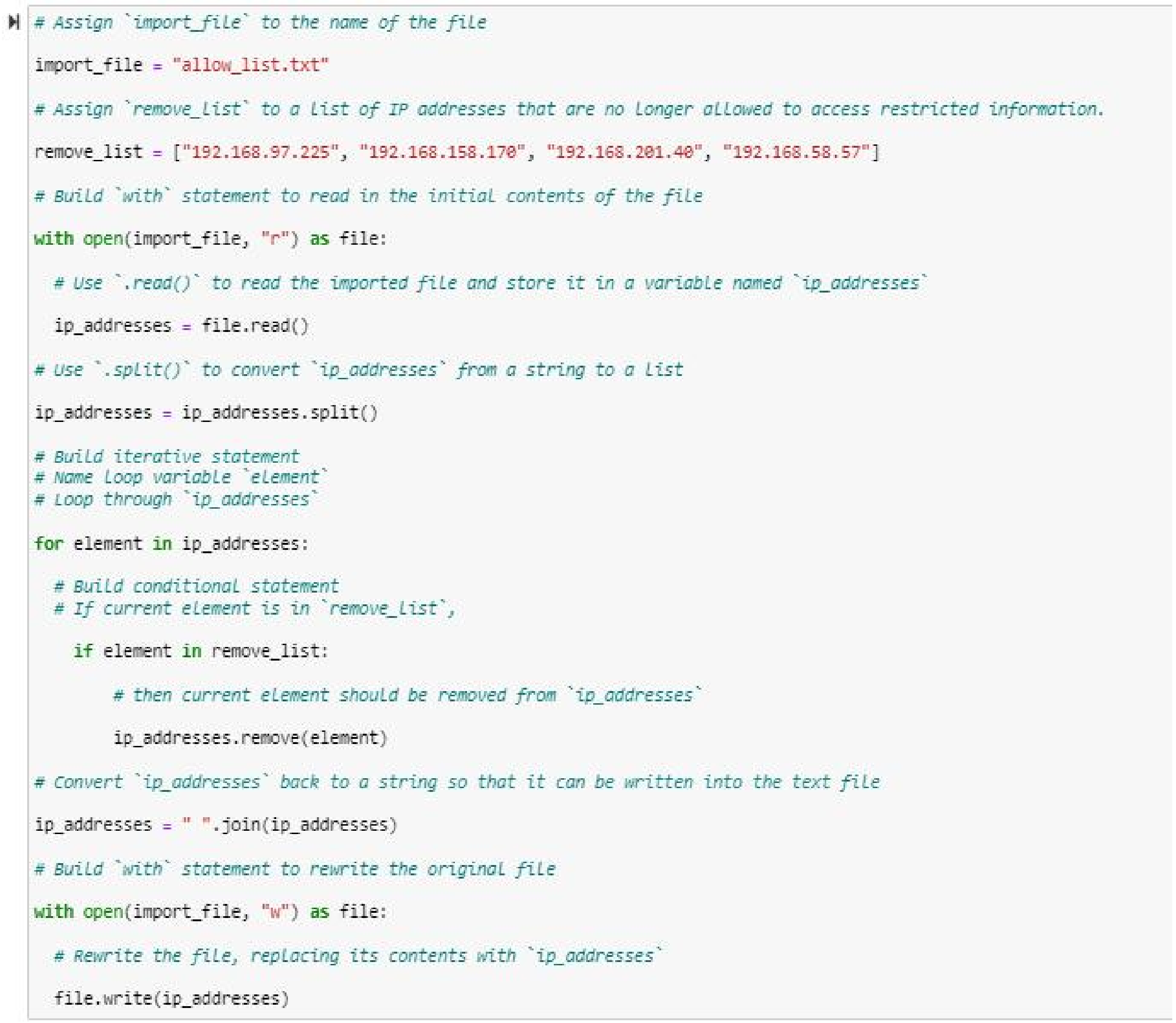


```
# Assign `import_file` to the name of the file

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)

# Display `ip_addresses`

print(ip_addresses)
```

```
['ip_address', '192.168.25.60', '192.168.205.12', '192.168.6.9', '192.168.52.90', '192.168.90.124', '192.168.186.176', '192.
168.133.188', '192.168.203.198', '192.168.218.219', '192.168.52.37', '192.168.156.224', '192.168.60.153', '192.168.69.116']
```

Update the file with the revised list of IP addresses

The **.join()** method is used with the separator "\n" to convert the list back into a string with IP addresses separated by new lines. Then, a **with** statement is used to open the file in write mode ("w") and the **.write()** method is used to write the updated content to the file.



```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()

# 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)
```

Summary

Project Description:

In the context of this scenario, I work as a security professional at a healthcare company, and part of my responsibility is managing access to restricted content based on IP addresses. Employees who are permitted access are listed in the "allow_list.txt" file. To maintain this list, I've developed a Python algorithm that can update it. This algorithm involves opening the file, reading its contents, converting the content into a list of IP addresses, iterating through another list of IP addresses to be removed, removing the specified IPs, and finally, updating the file with the revised list.

Summary:

The Python algorithm presented in this document serves a critical role in the security management of a healthcare company. By reading an initial list of allowed IP addresses from a file, converting it into a list for easier manipulation, and subsequently iterating through a removal list to eliminate IP addresses that should no longer have access, this algorithm automates a vital task. Its functionality is encapsulated in a **update_file** function, enhancing modularity and reusability. Moreover, it exemplifies the effective use of Python's file handling capabilities, including reading, manipulating, and rewriting file content. Ultimately, this algorithm offers an efficient solution to the challenge of maintaining a list of authorized IP addresses, streamlining the security operations within the healthcare organization.