Algorithm for file updates in Python

Project description

In this project, the goal was to automate the process of maintaining an allow list of IP addresses for a security system. This process involves reading an existing file containing a list of IP addresses, removing certain addresses that should no longer be allowed, and then updating the file with the revised list. The project utilized various Python functionalities, including file handling, string manipulation, and list operations, to achieve this.
The task begins with reading an allow_list.txt file containing a list of IP addresses, then removing IP addresses from the list that are specified in a remove_list. The updated list of IP addresses is then written back to the same file, ensuring that only the remaining allowed IPs are stored. Python's built-in functions like .read(), .split(), .remove(), .join(), and file handling techniques using with statements were employed to efficiently perform these tasks.

Open the file that contains the allow list

```
import_file = "allow_list.txt"    # Assigning the filename to the variable

# Using the 'with' statement to open the file safely
with open(import_file, 'r') as file:
    # Inside this block, you can work with the file
    contents = file.read()   # For example, read the contents of the file
    print(contents)    # Printing the contents of the file
```

Read the file contents

```
import_file = "allow_list.txt" # Assign the file name
 # Open the file using 'with' statement
with open(import_file, 'r') as file: ip_addresses = file.read() # Reads the entire content
of the file into a string
```

Convert the string into a list

```
import_file = "allow_list.txt"  # Assign the file name

# Open the file using 'with' statement
with open(import_file, 'r') as file:
    ip_addresses = file.read()  # Reads the entire content of the file into a string

# Convert the string into a list using the .split() method
ip_list = ip_addresses.split()  # Splits the string into a list of IP addresses
```

Iterate through the remove list

```
# Example lists for the task
ip_list = ["192.168.0.1", "192.168.0.2", "192.168.0.3"] # The list of IP addresses
 remove_list = ["192.168.0.2", "192.168.0.3"] # The list of IP addresses to remove
 # Header of the for loop to iterate through the remove_list
 for element in remove_list: # This is where you would implement the logic to remove
'element'
```

Remove IP addresses that are on the remove list

```
ip_list = ["192.168.0.1", "192.168.0.2", "192.168.0.3"]
# The list of IP addresses
remove_list = ["192.168.0.2", "192.168.0.3"] # The list of IP addresses to remove
# Iterate through each element in the remove_list
for element in remove_list: # Conditional to check if the element is in ip_list
if element in ip_list: ip_list.remove(element) # Remove the element from ip_list
```

Update the file with the revised list of IP addresses

```
# Step 1: Convert the ip_addresses list into a string, with each IP on a new line
updated_ip_addresses = "\n".join(ip_list)  # Join the IP list into a string with
newlines between each IP address

# Step 2: Use a with statement to open the file and write the updated string back to the
```

```
file
with open(import_file, 'w') as file:
    file.write(updated_ip_addresses)  # Write the updated IP list back into the file
```

Summary

The project involved reading a list of IP addresses from a text file, converting that data into a list format, and then removing specific IP addresses based on another list. After the removal of the specified IPs, the list was converted back into a string and written to the same file, updating the allow list. The project utilized key Python concepts such as list manipulation, file operations, and string processing.

The program demonstrated how automation can streamline security management tasks. By using Python, security analysts can easily modify the IP allow list based on predefined conditions, such as removing IP addresses associated with failed login attempts or identifying potential threats. The automation of these processes reduces the risk of human error and increases operational efficiency in maintaining an accurate and secure access list.

In summary, this project showcased Python's ability to automate critical aspects of cybersecurity, specifically in managing access control lists, with practical applications in various security systems that rely on IP address validation and access control.