# Algorithm for file updates in Python

## Project description

At my healthcare organization, access to restricted content containing sensitive patient records is controlled by an allowed list of IP addresses. As a security professional, I regularly update the `allow_list.txt` file to ensure that only authorized employees maintain access based on their current roles. I created a Python algorithm to automate the process of checking this allow list against a separate "remove list" of unauthorized IP addresses. This script removes identified addresses and updates the file to maintain a secure subnetwork.

## Open the file that contains the allow list

For the first part of the algorithm, I assigned the name of the file to a variable named `import_file`. Then, I used a `with` statement to open the file in read mode:

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

import_file = "allow_list.txt"
```

```
# Build `with` statement to read in the initial contents of the file

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

The `with` statement is used with the `open()` function to ensure that resources are managed properly by closing the file automatically after exiting the block. I used the `"r"` argument to indicate that I wanted to read the file.

## Read the file contents

To access the data within the file, I used the `.read()` method to convert the contents into a string format.

```
# Use `.read()` to read the imported file and store it in a variable named `ip_addresses`

ip_addresses = file.read()
```

This method takes the output of the opened file and stores it as a string in the variable `ip_addresses`, making it available for further manipulation within the Python program.

## Convert the string into a list

In order to remove individual IP addresses, the data needed to be in a list format. I used the `.split()` method to accomplish this:

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

ip_addresses = ip_addresses.split()
```

By default, the `.split()` function splits the text string by whitespace into individual list elements. Reassigning the result back to `ip_addresses` allows me to iterate through the addresses as elements in a sequence.

## Iterate through the remove list

To examine every IP address that needs to be removed, I implemented a `for` loop.

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

for element in remove_list:
```

The `for` keyword begins the loop, and the loop variable `element` represents each individual IP address in the `remove_list` as the algorithm iterates through the sequence.

## Remove IP addresses that are on the remove list

Inside the loop, I created a conditional statement to check if the current `element` was present in the `ip_addresses` list.

```
for element in remove_list:

  # Build conditional statement
  # If current element is in `ip_addresses`,

    if element in ip_addresses:

        # then current element should be removed from `ip_addresses`

        ip_addresses.remove(element)
```

I used the `.remove()` method because it successfully deletes a specific element from a list. This approach is effective here because there are no duplicate IP addresses within the `ip_addresses` list.

## Update the file with the revised list of IP addresses

After the removals, I needed to convert the list back into a string to write it back to the file. I used the `.join()` method.

```
# 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)
```

I used the `.join()` method to combine the list elements back into a single string. Then, I used a second `with` statement with the `"w"` argument to open the file in write mode, which overwrites the existing content with the updated string.

## Modularizing the algorithm into a function

To make the code reusable and organized, I wrapped the logic inside a function called `update_file` and called it with the relevant parameters.

```
# 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):

  with open(import_file, "r") as file:
    ip_addresses = file.read()

  ip_addresses = ip_addresses.split()

  for element in remove_list:
    if element in ip_addresses:

      ip_addresses.remove(element)

  ip_addresses = " ".join(ip_addresses)

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

By defining the function with parameters for the file name and the remove list, I can easily apply this logic to different files or address lists in the future. The final block of code reads the updated file and prints the content to the console to verify that the removals were successful.

## Summary

I created a Python algorithm that automates the removal of unauthorized IP addresses from an access control file. The process begins by opening the file and converting its string content into a list format for easier manipulation. By using a `for` loop and a conditional `if` statement, the algorithm iterates through a remove list and strikes matching addresses from the allow list. Finally, the `.join()` method converts the data back into a string so the `allow_list.txt` file can be updated with the revised permissions.