# Algorithm for file updates in Python

## Project description

I work as a security professional in a healthcare organization. As part of my responsibilities, I am required to regularly update a file that identifies which employees are allowed to access restricted content. Access to this content is controlled based on IP addresses, and the file contains a list of authorized IP addresses for employees who handle personal patient records.

Additionally, there is a removal list that identifies employees whose access should be revoked. My task is to create a Python algorithm that checks whether any IP addresses in the allow list are present in the removal list. If a match is found, the algorithm removes those IP addresses from the allow list file, ensuring that only authorized employees retain access to the restricted subnetwork.

## Open the file that contains the allow 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"]

# First line of `with` statement
with open(import_file, "r") as file:
```

I began by assigning the filename `"allow_list.txt"` as a string to the variable `import_file`. This specifies which file Python should work with. I also created a variable named `remove_list` containing the IP addresses that should no longer have access to restricted information.

To access the file's contents, I used the `with` statement together with the `open()` function. The `with` statement ensures the file is automatically closed after its block is executed, even if an error occurs. Inside `open()`, I passed two arguments: the variable `import_file` and the mode `"r"`, which stands for read mode. I used the keyword `as` to assign the opened file object to the variable `file`, allowing me to work with its contents inside the block.

The main Python syntax elements I used are:

- **Variable assignment (`=`)** to store the filename and the list of IP addresses.
- **The `with` statement** to manage file context.
- **The `open()` function** with the `"r"` mode to read the file.
- **The `as` keyword** to create a temporary reference (`file`) to the opened file object.

## Read the file contents

```
# 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)
```

Once the file was open, I used the `.read()` method on the `file` object to load the entire contents of `"allow_list.txt"` into memory. This method returns the data as a single string, which I stored in the variable `ip_addresses`. Having the data in a variable makes it easier to process without reopening the file.

After storing the contents, I used the `print()` function to display `ip_addresses` in the console, confirming that the file had been read successfully.

The main Python syntax elements I used are:

- **The `.read()` method** to retrieve the entire file content as a string.

- **Variable assignment (`=`)** to store the string in `ip_addresses`.
- **The `print()` function** to display the stored data in the console.

## Convert the string into a 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()

# Display `ip_addresses`
print(ip_addresses)
```

To work with individual IP addresses, I needed `ip_addresses` in list format rather than as a single string. I used the `.split()` method to break the string into separate elements, relying on its default behavior of splitting on whitespace. This works because each IP address in the file is separated by spaces or new lines.

I then reassigned the resulting list to the same variable, `ip_addresses`, replacing the original string. Finally, I used the `print()` function to display the updated list, verifying that the conversion was successful.

The main Python syntax elements I used are:

- **The `.split()` method** to convert the string into a list of elements.
- **Variable reassignment (`=`)** to store the new list in `ip_addresses`.

- **The `print()` function** to display the updated list in the console.

## 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)
```

To prepare for removing specific IP addresses, I set up a `for` loop to go through the `ip_addresses` list one element at a time. I used the loop variable name `element` to represent each individual IP address during each iteration.

Inside the loop, I used the `print()` function to display the current `element`. This allowed me to verify that the loop was cycling through each IP address in the list correctly before performing any removal operations.

The main Python syntax elements I used are:

- **The `for` statement** to iterate through the list sequentially.
- **A loop variable (`element`)** to store the current item from the list during each iteration.
- **The `print()` function** to display each IP address in the console.

## Remove IP addresses that are on 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:

  # 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)
```

To eliminate specific IP addresses from the allow list, I added a conditional statement inside the `for` loop. For each `element` in `ip_addresses`, the `if element in remove_list` condition checks whether the current IP address is included in the list of addresses to remove.

When the condition evaluates to `True`, I used the `.remove()` method on `ip_addresses` to delete the current `element` from the list. This approach works correctly because there are no duplicate IP addresses in `ip_addresses`, so each `.remove()` call only removes one unique occurrence.

Finally, I used the `print()` function to display the updated `ip_addresses` list, confirming that all addresses in `remove_list` were removed.

The main Python syntax elements I used are:

- **The `for` statement** to iterate through the list of IP addresses.
- **A loop variable (`element`)** to hold the current IP address during each iteration.
- **The `if` statement** to conditionally check membership in `remove_list`.
- **The `.remove()` method** to delete the current IP address from `ip_addresses`.
- **The `print()` function** to display the final list in the console.

## 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
remove_list = ["192.168.97.225", "192.168.158.170", "192.168.201.40", "192.168.58.57"]

# Read the initial contents of the file
with open(import_file, "r") as file:
  # Read file and convert it directly to a list of lines
  ip_addresses = file.read().splitlines()

# Remove any IPs that are in the remove_list (safe way)
ip_addresses = [ip for ip in ip_addresses if ip not in remove_list]
```

```
# Convert the list back to a string, separating each IP by a newline
ip_addresses_str = "\n".join(ip_addresses)

# Rewrite the original file with the updated list
with open(import_file, "w") as file:
  file.write(ip_addresses_str)

# Optional: display the updated IP addresses
print(ip_addresses_str)
```

---

After removing the unwanted IP addresses from `ip_addresses`, I converted the updated list back into a single string using the `"\n".join(ip_addresses)` method. This inserts a newline character between each IP address, ensuring that each entry appears on its own line in the file. I stored the resulting string in the variable `ip_addresses_str`.

Next, I used a `with` statement along with the `open()` function in write mode (`"w"`) to overwrite the original file assigned to `import_file`. Inside the `with` block, I applied the `.write()` method to save the contents of `ip_addresses_str` to the file. This approach guarantees that the file is automatically closed after writing, even if an error occurs.

Finally, I used the `print()` function to display the updated IP addresses, confirming that the file now contains the revised list.

The main Python syntax elements I used are:

- **The `"\n".join()` method** to convert the list into a single string with newlines between elements.
- **Variable assignment (`=`)** to store the joined string in `ip_addresses_str`.
- **The `with` statement** to safely manage the file context for writing.
- **The `open()` function** with mode `"w"` to open the file for writing.
- **The `.write()` method** to update the file with the new content.
- **The `print()` function** to display the updated IP addresses in the console.

## Summary

In this lab, I developed a Python algorithm to manage and update a list of allowed IP addresses. I started by opening the file containing the allow list and reading its contents into a variable. I then converted the data from a single string into a list to allow individual IP address manipulation. Using a `for` loop and conditional statements, I identified and removed IP addresses that were included in a separate removal list. Finally, I converted the updated list back into a string and wrote it to the original file, ensuring that the allow list was updated correctly. Throughout the process, I applied key Python concepts including file handling, list operations, loops, conditionals, and string methods, while ensuring the code remained readable and efficient.