Algorithm for file updates in Python

Project description

In this project, I developed a Python algorithm for a health care company to manage employee access to restricted content based on IP addresses. The algorithm updates an allow list of IP addresses by removing those found on a specified remove list. This ensures that only authorized employees retain access to personal patient records, aligning with security protocols.

Open the file that contains the allow list

To open the file containing the allow list, I used the `open()` function within a `with` statement. This ensures the file is properly managed and automatically closed after use. The syntax for opening the file is as follows:

```
import_file = "allow_list.txt"
with open(import_file, 'r') as file:
    ip_addresses = file.read()
```

In this code:

- `import_file` is assigned the name of the file to be opened.
- The `with` statement opens the file in read mode (`'r'`) and assigns the file object to the variable `file`.
- This allows the program to work with the file while ensuring it is closed after reading.

Read the file contents

To read the contents of the allow list file, I used the `.read()` method. This method retrieves all the text from the file as a single string, which is stored in the `ip_addresses` variable:

```
ip_addresses = file.read()
```

This step is crucial as it converts the file's contents into a string format that can be manipulated further in the algorithm.

Convert the string into a list

Since the IP addresses need to be managed as a list, I utilized the `.split()` method to convert the `ip_addresses` string into a list of individual IP addresses:

```
ip_addresses_list = ip_addresses.split("\n")
```

Here, the `.split("\n")` method divides the string at each newline character, creating a list where each element corresponds to a line in the original file.

Iterate through the remove list

To process the removal of IP addresses, I set up a for loop that iterates through a predefined list of IP addresses to be removed:

```
remove_list = ["192.168.1.1", "192.168.1.2"]
for element in remove_list:
```

This loop uses `element` as the loop variable, which represents each IP address in the `remove_list`.

Remove IP addresses that are on the remove list

Within the for loop, I implemented a conditional statement to check if each `element` from the `remove_list` exists in the `ip_addresses_list`. If it does, I used the `.remove()` method to delete it from the allow list:

```
    if element in ip_addresses_list:
        ip_addresses_list.remove(element)
```

This conditional checks for membership, and the `.remove()` method removes the first occurrence of the specified IP address from the list. This approach assumes no duplicates are present in `ip_addresses_list`.

Update the file with the revised list of IP addresses

After modifying the list of IP addresses, I converted the updated list back into a string format using the `.join()` method, separating each IP address with a newline:

```
updated_ip_addresses = "\n".join(ip_addresses_list)
with open(import_file, 'w') as file:
    file.write(updated_ip_addresses)
```

In this code:

- `"\n".join(ip_addresses_list)` creates a single string with each IP address on a new line.
- Another `with` statement opens the file in write mode (`'w'`) and writes the updated string back into the original file, effectively updating the allow list.

Summary

This algorithm effectively manages the security of employee access to sensitive data by maintaining an up-to-date allow list of IP addresses. Key components include opening and reading from the file, processing lists to manage access, and updating the file with the revised allow list. By employing Python's file handling capabilities and list operations, the solution ensures that only authorized personnel have access to restricted content, thus upholding the organization's commitment to data security.