Algorithm for file updates in Python

Project description

A health care organization tasked me with regularly updating a file that tracks employees authorized to access restricted content. This file contains the IP addresses of employees with access to personal patient records, and I was provided with a list of IPs to remove. For this project, I developed a Python algorithm that cross-checks the list of IP addresses to remove against the allow list, and eliminates any matches.

Open the file that contains the allow list

First I started by storing the **allow_list.txt** file path to a variable that I can use to open it.

```
import_file = "allow_list.txt"
```

I then used the **with open()** as file syntax to open the file to read the currently allowed IP addresses listed in the file.

```
with open(import_file) as file:
```

Read the file contents

Once the file was open I could read the contents with the **.read()** method which converts the contents of the file into a python string.

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

Convert the string into a list

Since I needed to iterate through the list of IP addresses to remove any unallowed IP addresses, I converted the file contents string into a python list using **.split()**.

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

ip_addresses = ip_addresses.split()
```

Iterate through the remove list

Once all the allowed IP addresses were in a list, I iterated through the IP addresses I needed to remove so that I could see if they existed in the currently allowed IP addresses.

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

for element in ip_addresses:
```

Remove IP addresses that are on the remove list

I then created a conditional to remove an IP address from the list of allowed IP addresses if it was currently allowed but should actually be removed.

```
if element in ip_addresses:

  # use the `.remove()` method to remove
  # elements from `ip_addresses`

    ip_addresses.remove(element)
```

Update the file with the revised list of IP addresses

With the updated list of IP addresses in hand, I had to update the **allow_list.txt** file. First, I converted the Python list back into a string, ensuring each IP address appeared on its own line. Next, I opened the **allow_list.txt** file using the **"w"** mode to overwrite its contents entirely and prevent any duplicates. Finally, I wrote the updated list of allowed IP addresses to the file.

```
ip_addresses = "\n ".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

For this task, a Python algorithm was developed to process a file of IP addresses and update it using a list of IP addresses to be removed. By leveraging Python functionalities such as **with open**, **.read()** and **.remove()**, the file was converted into a Python list. The algorithm then iterated through the list of IP addresses to remove, comparing them to the ones in the file. After removing the invalid IP addresses, the updated list was converted back into a string, allowing the valid IP addresses file to be refreshed with the new list.