Algorithm for file updates in Python

Project description

My healthcare company has a security control set in place with an allow list containing IP addresses authorized to have access to restricted content. Each employee has a designated IP address. The allow list file is aptly named `"allow_list.txt"`, which identifies these authorized IP addresses. A separate remove list identifies IP addresses that are no longer authorized to have access to this restricted information. I developed an algorithm that automates updating the `"allow_list.txt"` file by removing the IP addresses identified in the remove list through a filter.

Open the file that contains the allow list

For the first part of my algorithm, I assigned the `"allow_list.txt"` string to the `import_file` variable. I then assigned the `["192.168.97.225", "192.168.158.170", "192.168.201.40", "192.168.58.57"]` list to the `remove_list` variable, that will be used to update the `allow_list.txt` file. Next, I used the `with` statement to open my import file and store it as a variable `file` to be used as a reference to the imported file, within the `with` statement:

```
In [11]: # 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:
```

- The `with` statement is used for file handling. It closes the open file automatically after the `with` statement exits.
- Python's built-in `open()` function takes in two arguments: the first argument takes in a string or variable containing the relative or absolute path of the file to be opened, and the second argument takes in the file operation as a string (I.e. `"r"`, `"w"`, `"a"`).
    - The `"r"` argument reads the open file. To convert a file into a string, the `.read()` method is used on the file object.

    - The `"w"` argument writes to the open file. To convert a string back to the file, the `.write()` method is used on the file object. This operation can be used either to overwrite an existing file or to create a new file.
    - The `"a"` argument appends to the end of the open file. To convert and append a string to the end of the open file, the `.write()` method is used on the file object. This operation does **not** overwrite existing files.
- The `as` portion of the `with` statement specifies what variable name we want to use, within the indented portion after the `with` statement, to refer to the open file. For example, using the `with open(import_file, "r") as file:` header, we know that the variable `file` will be used as the variable name that refers back to the open file.

Read the file contents

In order to read the contents from the allow list, I used the `.read()` method that is part of the file object.

```
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()
```

To properly read the contents of the imported file, I placed the `“r”` as the second argument to Python’s built-in `open()` function. This argument also allows us to use the `.read()` method to convert the `"allow_list.txt"` file to a string. Once the file is converted to a string, I then placed the string into the `ip_addresses` variable.

As a string, I can now organize and extract data from the file more seamlessly in my Python program.

Convert the string into a list

Before I remove the IP addresses from the `"allow_list.txt"` file, I need to convert the `ip_addresses` variable from a string into a list. I use the `.split()` method to achieve this conversion:

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

ip_addresses = ip_addresses.split()
```

The `.split()` method is a function belonging to the string data type; in this case, it belongs to the `ip_addresses` variable. The purpose behind converting this string to a list is to make IP address removal from the allow list easier to implement. The argument within the `.split()` method takes in a separator character that specifies which character to split on. Each segment between split characters becomes an element in the list. If a character is not specified, the method will split on whitespace by default. Since the string containing the IP addresses is separated by whitespace, we don't specify an argument and instead use the default whitespace separator. To store the result as a list, I reassigned it back to the `ip_addresses` variable.

Iterate through the IP address list & filter out IP addresses that are on the remove list

In order to check which IP addresses need to be removed from the allow list, my algorithm iterates through `ip_addresses` using list comprehensions:

```
# Build list comprehension
# Name loop variable `element`
# Loop through `ip_addresses
# Build conditional statement
# Exclude current element from new list if it is within the `remove_list` variable

ip_addresses = [element for element in ip_addresses if element not in remove_list]
```

A list comprehension is a concise, efficient and pythonic way to create lists in Python. It allows us to generate a new list by iterating over an iterable, such as another list, and applying a condition or transformation to each element. In this case, we apply a condition to filter through the `ip_addresses` by checking if the current element matches any element within the `remove_list` variable.

In the `for element in ip_addresses` portion of the code, the `for` keyword starts the for loop, the `in` keyword tells us that we will iterate over all the elements within the `ip_addresses` variable and assign each element to the `element` loop variable for each iteration. For each element, we apply a condition: `if element not in remove_list`. This is the condition that filters the elements. It checks if the current element is `not in` the `remove_list`. If the condition evaluates to `True`, the element is included in the new list; otherwise, it is excluded. We then reassign this new list back to the `ip_addresses` variable.

Update the file with the revised list of IP addresses

Lastly, my algorithm updates the allow list using the revised `ip_addresses` variable. Before updating the allow list, I needed to convert the `ip_addresses` variable back to a string using the `.join()` method:

```
# Convert `ip_addresses` back to a string so that it can be written into the text file
ip_addresses = "\n".join(ip_addresses)
```

The `.join()` method combines an iterable, such as a list, into a single string. This method is appended to the string that we want to separate the iterable by once it's converted into the single string we mentioned. In this case, my algorithm converts the `ip_addresses` iterable to a single string, and I use a `"\n"` separator, so this string displays each element, or IP address, on a new line. It performs this conversion by passing the iterable as an argument to the `.join()` method, so that I can then pass the resulting string as an argument to the `.write()` method when writing to the `"allow_list.txt"` file.

Next, I used a second `with` statement and the `.write()` method to update the allow list file:

```
# 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)
```

As opposed to the first `with` statement, I chose "w" as the second argument to the `open()` function. This argument enables us to overwrite an existing file or create a new file using the `.write()` method. In this case, we are overwriting the `"allow_list.txt"` file.

I appended the .`write()` method to the `file` object that I created in the `with` header. The `.write()` method takes an argument that contains the string contents used to overwrite the file contents specified in the `with` statement. In this case, I used the revised `ip_addresses` variable as the argument to the `.write()` method. Once this operation is finished, the restricted content will no longer be accessible to the IP addresses that were removed from the allow list.

Summary

To conclude, I created an algorithm in Python that filters out, and effectively removes, IP addresses that were identified in the `remove_list` variable from the `“allow_list.txt”` file. The algorithm started by opening the file, converting it to a string to make it readable, and then converting this string to a list which was stored in the `ip_addresses` variable. Next, I iterated through the `ip_addresses` variable using list comprehensions. Each iteration checked whether the current element was in the `remove_list` through an `if` condition. If this condition evaluated to `True`, I excluded the element from the revised IP addresses list. If the condition evaluated to `False`, I included the element to the revised list. Then, I used the `.join()` method to convert the `ip_addresses` list back to a string, so I could easily overwrite the outdated contents of the `"allow_list.txt"` file with the updated IP addresses using the `.write()` method.