Algorithm for file updates in Python

Project description

In order for organizations to make sure that only authorized individuals have access to restricted content, they often employ the use an allow list for IP addresses. In this document, I will demonstrate how Python can be utilized to create an algorithm which automates the process for updating an `"allow_list.txt"` file by removing IP addresses that should no longer have access to restricted content.

Open the file that contains the allow list

The first step to creating the algorithm is opening the `"allow_list.txt"` file in Python. I begin by assigning the file name as a string to the `import_file` variable and opening it by using a `with` statement:

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

import_file = "allow_list.txt"

# Create a `with` statement to read the current contents of the file

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

Here, the `with` statement is used with the `open()` function in read mode to open the file containing the allow list. This way I can access the IP addresses stored in the allow list file. Utilizing the `with` keyword is more effective for resource management because it closes the file after exiting the `with` statement. The first parameter in the `open()` function selects the file that I wish to import and the second indicates what will be done to the file – `"r"` meaning that it will be read. I also assign another variable named `file` which stores the output of the `open()` function while within the `with` statement. This is achieved with the help of the `as` keyword.

Read the file contents

The contents of the file should then to converted into a string, so that I can read them in Python. I do this by employing the `.read()` method:

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

  # Use `.read()` to read the imported file whilst also storing it in a variable named `ip_addresses`

  ip_addresses = file.read()
```

The argument `"r"` that I included in the `open()` function allows for the use of the `.read()` method in the body of the `with` statement. I convert the data from the .TXT file into a string that I will be able to read. I apply the `.read()` method to the `file` variable that was identified in the header and assign the output to a new variable called `ip_addresses`. I am now ready to organize and extract data from this string in Python.

Convert the string into a list

For me to be able to remove individual IP address from the allow list, I need to convert the data once again. This time from string to list format. This is done by using the `.split()` method on the `ip_addresses` variable that I created earlier:

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

ip_addresses = ip_addresses.split()
```

By default, the `.split()` method splits the text at every white space into a list of elements. In the variable `ip_addresses`, the data is currently a string of IP addresses where each one is separated by a whitespace. This means that I do not need to add any arguments to the method in order to achieve the desired result. The output of this is a list of IP addresses that I reassigned back to the variable `ip_addresses`.

Iterate through the remove list

Now comes the main part of the algorithm where I it will iterate through another list of IP addresses that are stored in a variable called `remove_list`. This is achieved by creating a `for` loop which begins as follows:

```python
#Create an iterative statement that loops through 'remove_list'
#Assign a loop variable called 'element'

for element in remove_list:
```

This loop repeats code for every element in `remove_list`. It starts with the keyword `for` which is then followed by the loop variable `element`. This is then followed by the keyword `in` which indicates to iterate through the `remove_list` to assign each value to the `element` loop variable.

Remove IP addresses that are on the remove list

The algorithm needs to remove the IP addresses that are in `remove_list` from the `ip_addresses` variable which contains the allow list. Since no duplicate addresses are present in `ip_addresses`, the following code is enough to complete this task:

```
for element in remove_list:

  # Nest conditional statement to evaluate if 'element' is present in the'ip_addresses' list

    if element in ip_addresses:

        # Use 'remove()' method to remove current element from `ip_addresses`

        ip_addresses.remove(element)
```

This was achieved by creating a nested conditional within the loop which evaluates whether the loop variable `element` is found on the list contained in `ip_addresses`. This was done in this particular way because applying `.remove()` to elements that are not present in ip_addresses would result in an error message. Applying `.remove()` within the conditional with the loop variable `element` as the argument makes sure that each IP address that is in `remove_list` would be removed from the `ip_addresses` allow list.

Update the file with the revised list of IP addresses

In order to finalize the algorithm, I need to update the original allow list .TXT file with the revised list of IP addresses. This is done by reversing the process I used to open and edit the file in Python. It needs to first be converted back from a list into a string. This can be done with 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)
```

Utilizing `.join()` combines all items from a list into a string. The string that gets created is then reassigned to the `ip_addresses` variable so that it could then be passed as an argument argument to the `.write()` method when overwriting the `"allow_list.txt"` file. In this particular case, I used the string `("\n")` as the separator which instructs Python to place each element from the list onto a new line in within the string. However, this could also be achieved by using `(" ")` as the separator. They both count as whitespace, so either will work, but I went with `("\n")` for readability.

I then once again used a `with` statement, but this time with the `.write()` method in order to update the 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)
```

By using `"w"` as the second argument of the `open()` function in the my `with` statement, I am letting Python know that I want to open a file and write over its contents. This allows me to call the `.write()` method in the body of the `with` statement which then writes the data from the string onto a specified file, replacing its previous contents. The `.write()` method is appended to the file object named `file` which was indicated in the header of the `with` statement. By specifying the `ip_addresses` variable as the argument, I am telling Python to replace the contents of the file in the `with` statement with the data string stored in the variable.

Now the allow list of the organization is updated so that restricted content is no longer accessible to any of the IP addresses that were on the remove list. This code could also be incorporated into a custom function, so that next time only the allow list .TXT file and the remove list are given to the algorithm and it automatically updates the list of approved IP addresses.

Summary

This project demonstrated how Python can be used to create an algorithm that removes a specified list of IP addresses, stored in a variable named `remove_list`, from an allow list .TXT file. The algorithm works by opening up the `"allow_list.txt"` file, converting it to a readable string, and then transforming this string into a list stored in a new variable named `ip_addresses`. A `for` loop is then used to iterate through the IP addresses in `remove_list` and match them to equivalent elements in the `ip_addresses` list. If there is a match, the `.remove()` method is applied to remove the corresponding unauthorized IP address from the list. Afterwards, I utilize the `.join()` method to convert the `ip_addresses` list back into a string, so that the contents of the original `"allow_list.txt"` can be overwritten with the `.write()` function.