Algorithm for file updates in Python

Project description

As a security professional working at a health care company, part of my job is to regularly update a file that identifies the employees who can access restricted content. The contents of the file are based on who is working with personal patient records. Employees are restricted access based on their IP address. There is an allow list for IP addresses (found in the `“allow_list.txt”`) permitted to sign into the restricted subnetwork. This is a documentation of an algorithm that uses Python code to check whether the allow list contains any IP addresses identified on the remove list. If so, I remove those IP addresses from the file containing the allow list.

Open the file that contains the allow list

For the first part of the algorithm, I opened the `"allow_list.txt"` file, assigned this file name as a string to the `import_file` variable and reused the `with` statement to open the file:

```python
# 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 r

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` keyword, along with the `.open()` function used with the read mode is used to open the allow list file for the purpose of reading it. I did this to access the IP addresses stored in the allowed list file. The `with` keyword is essentially as it opens and then closes the files after exiting the statement. In the code `with open(import_file, "r") as file:`, the `open()` function has two parameters: the first identifies the file to import, and then the second indicates what I want to do with the file (`"r"` indicates that I want to read it). The code also uses the `as` keyword to assign a variable named `file`; `file` stores the output of the `.open()` function while I work within the `with` statement.

Read the file contents

To read and print the file contents, I used the `.read()` method to convert it into the string.

```
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)
```

I applied the `.read()` method to the `file` variable identified in the `with` statement. Then, I assigned the string output of this method to the variable `ip_addresses`. I can call the `.read()` function in the body of the `with` statement which converts the file into a string and allows me to read it.

Convert the string into a list

To remove IP addresses from the allow list, I need it to be in a list form which is why I use the `split()` method to convert `ip_addresses` from 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 restr

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)
```

The `.split()` function converts the content of a string into a list. Right now, it converts the string `ip_addresses` into a list. This will make it easier for me to remove the IP addresses from the allow list. Since nothing was placed between the parentheses in the `.split()` function, it separated the string by whitespaces into the list elements. This is then restored back to the `ip_addresses`.

Iterate through the remove list

I then iterate through the IP addresses in the `remove_list` using a for loop. The purpose of the `for` loop in my algorithm is to apply specific code statements to all elements in a sequence. The `for` keyword starts the `for` loop. It is followed by the loop variable `element` and the keyword `in`. The keyword `in` indicates to iterate through the sequence `ip_addresses` and assign each value to the loop variable `element`.

```
# Build iterative statement
# Name loop variable `element`
# Loop through `ip_addresses`

for element in ip_addresses:

    # Display `element` in every iteration

    print(element)
```

Remove IP addresses that are on the remove list

The goal of my algorithm is to remove any IP address from the allow list, `ip_addresses`, that is also contained in `remove_list`. Since there were not any duplicates in `ip_addresses`, I was able to use the following code to do this:

```
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)
```

I created a conditional that evaluated whether or not the loop variable element was found in the `ip_addresses` list. I did this because applying `.remove()` to elements that were not found in `ip_addresses` would result in an error. Then, within that conditional, I applied `.remove()` to `ip_addresses`. I passed in the loop variable element as the argument so that each IP address that was in the `remove_list` would be removed from `ip_addresses`.

Update the file with the revised list of IP addresses

The final step in my algorithm was to update the allow list file with the revised list of IP addresses (after removing the duplicate IP addresses). I first needed to convert the list back into a string and I used the `.join()` method for this. The `.join()` method combines all items in an iterable into a string and is applied to a string containing characters that will separate the elements in the iterable once joined into a string. In this algorithm, I used the

`.join()` method to create a string from the list `ip_addresses` so that I could pass it in as an argument to the `.write()` method when writing to the file `"allow_list.txt"`. I used the string (`"\n"`) as the separator to instruct Python to place each element on a new line.

Then, I used another with statement and the `.write()` method to update the file with the second argument of "`w`" with the `open()` function in my with statement. This argument indicates that I want to open a file to write over its contents. When using this argument "`w`", I can call the `.write()` function in the body of the with statement. The `.write()` function writes string data to a specified file and replaces any existing file content. In this case I wanted to write the updated allow list as a string to the file `"allow_list.txt"`. This way, the restricted content will no longer be accessible to any IP addresses that were removed from the allow list. To rewrite the file, I appended the `.write()` function to the file object file that I identified in the with statement. I passed in the `ip_addresses` variable as the argument to specify that the contents of the file specified in the with statement should be replaced with the data in this variable.

```
# Convert `ip_addresses` back to a string so that it can be written into the text file

ip_addresses = " ".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

I created an algorithm that removes IP addresses identified in a `remove_list` variable from the `"allow_list.txt"` file of approved IP addresses. This algorithm involved opening the file, converting it to a string to be read, and then converting this string to a list stored in the variable `ip_addresses`. I then iterated through the IP addresses in `remove_list`. With each iteration, I evaluated if the element was part of the `ip_addresses` list. If it was, I applied the `.remove()` method to it to remove the element from `ip_addresses`. After this, I used the `.join()` method to convert the `ip_addresses` back into a string so that I could write over the contents of the `"allow_list.txt"` file with the revised list of IP addresses.