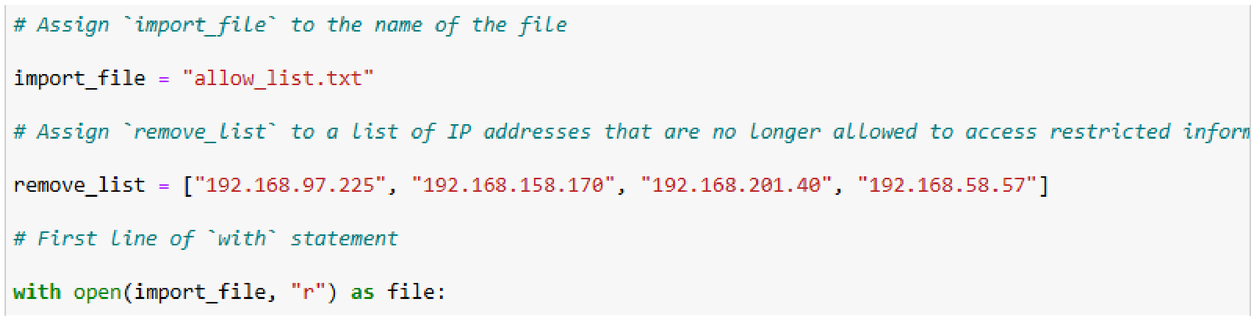
Algorithm for file updates in Python

Project description

This project was built to demonstrate my aptitude on parsing files as well as creating algorithms using Python. In this lab, we will cover many different functions which will ultimately build an algorithm which will help us in our hypothetical scenario as an security analyst.

Open the file that contains the allow list

We are tasked with the following: "The file that you want to open is called **`"allow_list.txt"`**. Assign a string containing this file name to the **`import_file`** variable. Then, use a **`with`** statement to open it. Use the variable **`file`** to store the file while you work with it inside the **`with`** statement."



```
# 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 inform

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:
```

Here the Syntax are as follows: Here we are assigning a name to a file which we have here as `import_file`. With this we move on to assigning a name for a list of IP addresses which we have here as `remove_list`. The use of with statement opens a resource and will close said resource when block is completed. Next we move onto the `open()` statement which require 2 parameters: The item in which we are calling and the argument. As for this example, we are calling for `import_file` and placing a `"r"` for Read. Followed by `as file` to consider `with open` statement as a file.

Read the file contents

For this example we are tasked with: use the **`.read()`** method to convert the contents of the allow list file into a string so that you can read them. Store this string in a variable called **`ip_addresses`**

```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 restricted infor

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()

# Display `ip_addresses`

print(ip_addresses)
```

We are to use the `.read()` statement to call upon the previous command and read it while calling it `ip_addresses`. Since we are now naming a new command we have to place that before the equals sign and because we are opening the file under the name `file` we should place the `read()` statement with that name. Finally we should print the output by calling ip_addressess hence `print(ip_addresses)`

Convert the string into a list

For this task we must: remove individual IP addresses from the allow list, the IP addresses need to be in a list format. Therefore, use the **`.split()`** method to convert the **`ip_addresses`** string into a list

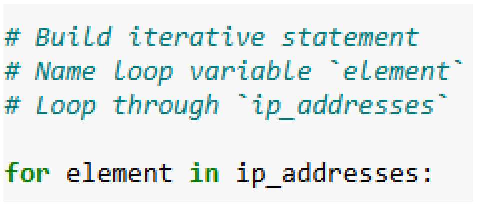
```python
# Assign `import_file` to the name of the file

import_file = "allow_list.txt"

# Assign `remove_list` to a list of IP addresses that are no longer allowed to access restricted info

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()` statement will allow us to separate each instance to a more readable format as a string. In further detail, the function will convert the contents of a string into a list. We can do this by adding an additional command with `ip_addresses = ip_addresses.split()` and then printing the output. To store this list, I was able to assign it back to the variable `ip_addresses`.

Iterate through the remove list

In the following we have: A second list called **`remove_list`** contains all of the IP addresses that should be removed from the **`ip_addresses`** list. Set up the header of a **`for`** loop that will iterate through the **`remove_list`**. Use **`element`** as the loop variable

This task requires both an iterative statement and a conditional statement. For this algorithm, the iterative statement would be the IP addresses which are elements within the remove_list.



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

for element in ip_addresses:
```

For this task I have used the `for ` loop because of its ability to apply specified code statements to all elements in a sequence,. The for loop will begin with `for` and then followed by the loop variable which is `element`. Then we have to specify the keyword `in` followed by the `ip_addresses` to specify the `for` loop to reiterate through.

Remove IP addresses that are on the remove list

For this algorithm, I would like to remove any IP addresses that are in the allow_list which should be removed. More specifically the allow_list and the remove_list will be compared and any IP address that matches shall be removed. To accomplish 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)

# Display `ip_addresses`

print(ip_addresses)
```

First we must justify why we have a conditional statement. This is because we have to evaluate whether or not the loop variable element was found in the ip address. If not done, the .remove() application would result in an error. Then within that conditional statement i was able to apply the .remove() command to ip_addresses. This loop will repeat until the contents within the ip_addresses and remove_list has been satisfied.

Update the file with the revised list of IP addresses

The final part of this algorithm requires the allow_list to be updated with the revised ip_addresses. For this we need 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()` statement combines all items into a string. After `using ,join()` we can pass this argument to the `.write()` method when writing to the file "allow_list.txt". The `("\n")` was used to instruct python to place each element onto a new line.
With this we are able to use another with statement an the `.write()` method to update the file

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

  # Rewrite the file, replacing its contents with `ip_addresses`

  file.write(ip_addresses)
```

The reason for this is the second argument of `"w"` with the `open()` function in my with statement indicates that I want to open a file to write over its contents. When using the argument `"w"`, I will be able to call upon the `.write()` function in the body of the with statement. This will allow us to write string data to a specified file and replace any existing file content.

Before, we were able to rename the `“allow_list.txt”` file into `import_file` which this algorithm will be overriding. This algorithm will be able to restrict access to any sensitive data from IP addresses that were removed from the allow list. The secondary `.write()` command will replace the data within the variable and output a new list with updated contents.

Summary

In conclusion, I have created an algorithm that removes IP addresses identified in a flagged list. This list was used to compare from the allow list and updated a new list. This algorithm involved opening the file, converting it to a string, and then converting this string to a list which was used as an iterative statement. It was pushed through a list of IP addresses contained in a different variable which evaluated each element of the original list. It applied the `.remove()` method to remove any element that matched and outputted an updated `allow_list` by using the `.join()` command.