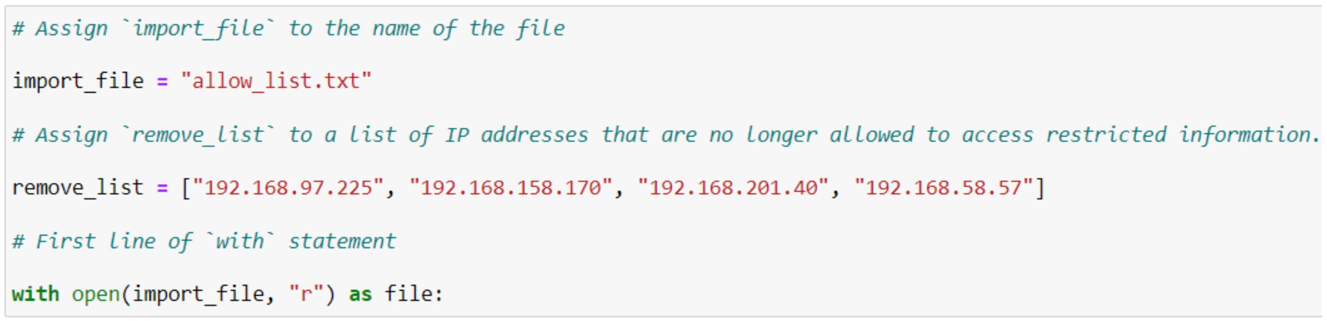
Algorithm for file updates in Python

Project description

This project simulates security tasks that require regularly updating a file regarding which employees have access to a restricted content. Employees are restricted access based on their IP address. There is an allow list under a file titled “allow_list.txt” and a remove list of IP addresses to work with. To automate this process, I will create an algorithm to check if any IP addresses in the remove list are in the allow list, and if so, remove them.

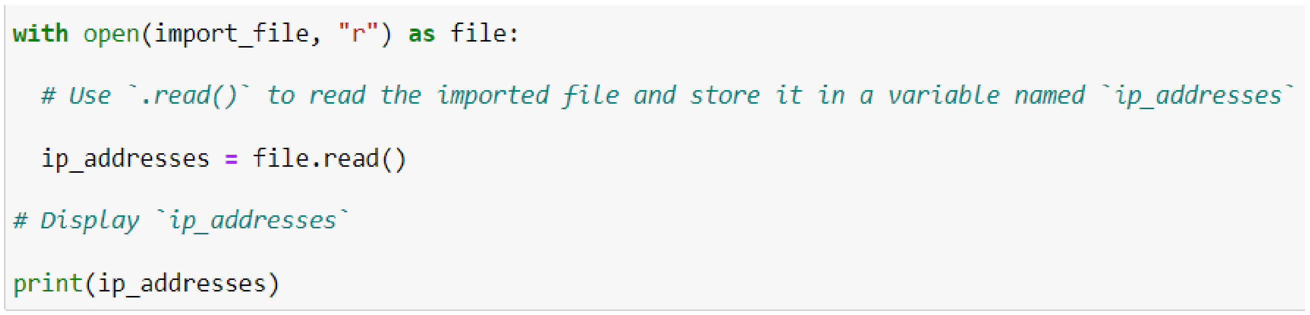
Open the file that contains the allow list



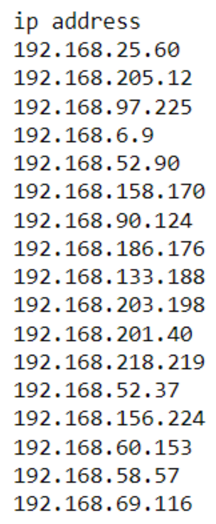
```
# Assign `import_file` to the name of the file

import_file = "allow_list.txt"

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

This is the start of opening the file with the allow list. Assigning the file itself (“allow_list.txt”) to the variable import_file will allow me to streamline the code and use the variable whenever it is needed later on. The same applies to the list of IP addresses being assigned to the variable remove_list. Reading files in Python requires the use of the with statement, and the open() function.

Read the file contents



```
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)
```



```
ip address
192.168.25.60
192.168.205.12
192.168.97.225
192.168.6.9
192.168.52.90
192.168.158.170
192.168.90.124
192.168.186.176
192.168.133.188
192.168.203.198
192.168.201.40
192.168.218.219
192.168.52.37
192.168.156.224
192.168.60.153
192.168.58.57
192.168.69.116
```

To read file contents in Python, the .read() method, with statement, and open() function are needed. I use the arguments import_file and “r” because I want to only read the contents in the file “allow_list.txt”. Since the file is assigned to the variable import_file, I did not have to type the file name itself for the argument. I print the output to see the contents of the “allow_list.txt” file to see what was in the file before editing it.

Convert the string into a list

```
# 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()
```

To make the string editable, it first has to be converted to a list. To accomplish this, the .split() method is needed. This method splits string data into a list of strings based on any whitespace characters present (in this case, the new line whitespace in the previous screenshot). I then assign the output from ip_addresses.split() to the variable ip_addresses.

Iterate through the remove list

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

for element in remove_list:
```

Removing the IP addresses from remove_list from the newly created ip_addresses list can be performed with an iterative statement, AKA a for loop. I make the for loop variable element and have the loop iterate through remove_list.

Remove IP addresses that are on the remove list

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

for element in remove_list:

    # Create conditional statement to evaluate if `element` is in `ip_addresses`

    if element in ip_addresses:

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

        ip_addresses.remove(element)
```

I will introduce a conditional statement in the for loop to remove all of the items from element from ip_addresses. For removing the items, the .remove() method is needed. This method removes the first occurrence of an element from the list it is removing from. The for loop will iterate through each element and will break out of the loop once complete.

Update the file with the revised list of IP addresses

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

# Build `with` statement to read in the updated file

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

    # Read in the updated file and store the contents in `text`

    text = file.read()

# Display the contents of `text`

print(text)
```

```
ip
address
192.168.25.60
192.168.205.12
192.168.6.9
192.168.52.90
192.168.90.124
192.168.186.176
192.168.133.188
192.168.203.198
192.168.218.219
192.168.52.37
192.168.156.224
192.168.60.153
192.168.69.116
```

To update the file with the revised list, the `.join()` method, `with` statement, `open()` function, `.write()` method, and `.read()` method are needed. Files can only be edited with string data, not list data. To make the list readable in string form, I assign the output of `“\n”.join(ip_addresses)` to the `ip_addresses` variable. I use `“w”` as the argument and the `.write()` method because I do not want to append the new content to the old content in the file, I want to rewrite the file’s content. To ensure the code was successfully performed, I open `import_file` and assign that output to the `text` variable, then print the output.

Summary

If there were hundreds of IP addresses, doing file updating manually would be a tedious task. This algorithm shows Python's capabilities in automating daily cybersecurity tasks. The algorithm uses an iterative and conditional statement to solve the problem. The `.read()`, `.write()`, `.split()`, and `.join()` methods are used in conjunction with the statements. Assigning variables makes writing code easier, so that I can grab large amounts of data from a small title whenever said large amounts of data is needed. Taking some time to create an algorithm in Python will save hours of manual work for the team when updating files.