Erick Tafel

Algorithm for file updates in Python

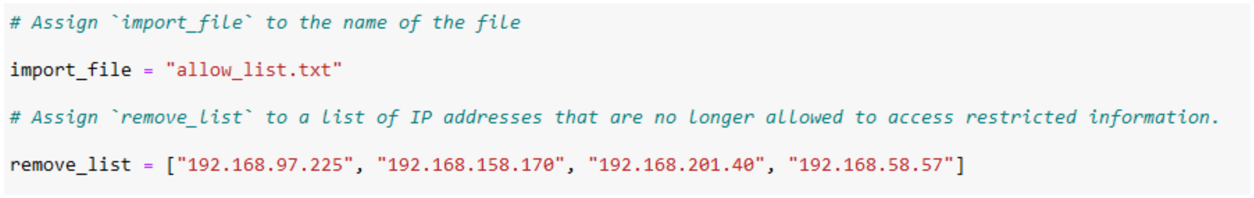
Project description

I am a security professional working at a health care company. I have been tasked with regularly updating 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 permitted to sign into the restricted subnetwork (`”allow_list.txt”`). There's also a remove list that identifies which employees you must remove from this allow list (`remove_list`).

Given this information, I created an algorithm that uses Python code to automate the checking of the `”allow_list.txt”` against any IP addresses identified on the `remove_list`. If the algorithm identifies removed addresses in the allow list, it is programmed to remove those IP addresses from the file containing the allow list. Below are the steps and screenshots of how I created this algorithm.

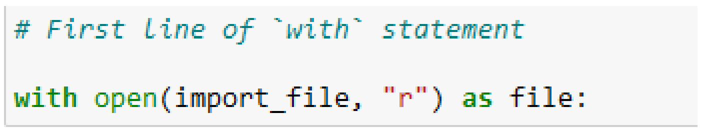
Open the file that contains the allow list

The beginning step was to open the `"allow_list.txt"` file. To do this, I first assigned the `"allow_list.txt"` file as a string to the `import_file` variable. I then assigned the list of IP addresses that are no longer allowed to access restricted information to the `remove_list` variable. Please see the screenshot below for reference:



```
# 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"]
```

Now that the `import_file` and `remove_list` is assigned, I then was able to open the `import_file` to read. To do this, I used a `with` statement with the parameters `import_file` (specifies the file to open), and `“r”` (allows me to read the file selected) and saved the open file `as file` (specifies what name to save the open file as). Please see the screenshot below for reference:

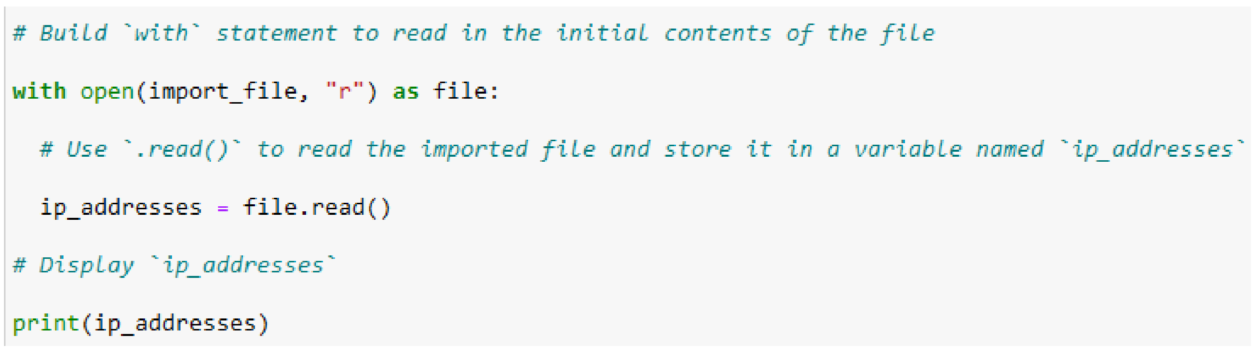


```
# First line of `with` statement

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

So far, the algorithm I created uses the `with` statement in conjunction with the `open()` function. The parameters inside of the `open()` function being the allow list file (`import_file`) and the reading mode (`“r”`). With the file now open, I can access the allow list IP addresses that are stored within, allowing me to use that data. It is important to note that with the use of the `with` keyword, the file will close after exiting the `with` statement. This is good practice and will help to manage resources. Referring to the screenshot before, we can see the full line of code: `with open(import_file, "r") as file:`, the code uses the `as` keyword to assign a variable to the open file, in this case I named the file `file`. The output of the `open()` function is stored here.

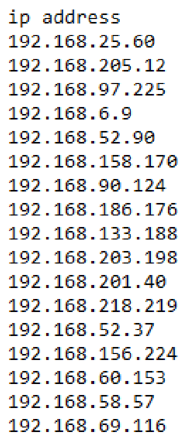
Read the file contents

The next step is to read the file contents that we opened. To do this, I used the `.read()` method. The `.read()` method converts the file into a string. I saved the converted output in a variable named `ip_addresses` and printed the output to confirm the code is functioning properly. Please see the screenshots below for reference:



```
# 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)
```

Output of `print(ip_addresses)`:

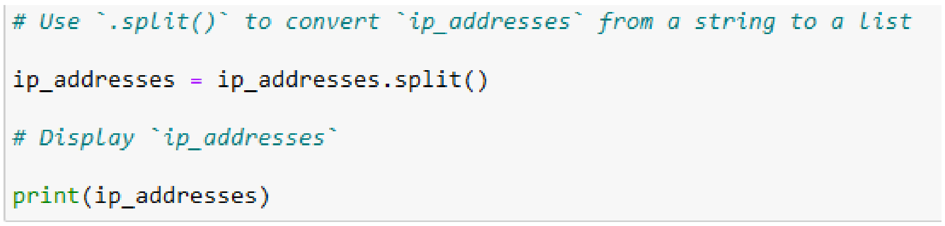


```
ip address
192.168.25.60
192.168.205.12
192.168.97.225
192.168.6.9
192.168.52.90
192.168.158.170
192.168.90.124
192.168.186.176
192.168.133.188
192.168.203.198
192.168.201.40
192.168.218.219
192.168.52.37
192.168.156.224
192.168.60.153
192.168.58.57
192.168.69.116
```

Thus far, the algorithm reads the contents of the `"allow_list.txt"` file into a string type. This allows me to modify the string in later steps.

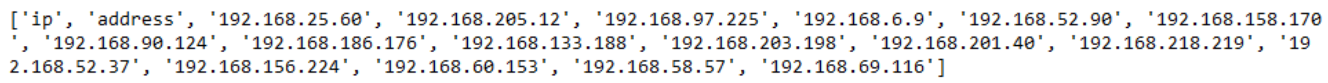
Convert the string into a list

The next step is converting the string into a list. This will help me to directly edit the allow list IP addresses. To do this, I used the `.split()` method and assigned the new output back to the variable `ip_addresses`. This method converts the specified variable (`ip_addresses`) string into a list. I printed the output to confirm the code is functioning properly. Please see the screenshots below for reference:



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

ip_addresses = ip_addresses.split()

# Display `ip_addresses`

print(ip_addresses)
```

Output of `print(ip_addresses)` after the file was converted to a list:

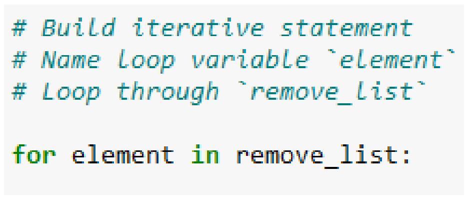


```
['ip', 'address', '192.168.25.60', '192.168.205.12', '192.168.97.225', '192.168.6.9', '192.168.52.90', '192.168.158.170
', '192.168.90.124', '192.168.186.176', '192.168.133.188', '192.168.203.198', '192.168.201.40', '192.168.218.219', '19
2.168.52.37', '192.168.156.224', '192.168.60.153', '192.168.58.57', '192.168.69.116']
```

My intention behind splitting `ip_addresses` into a list is to make the removal of IP addresses from the allow list easier. The way the `.split()` function works by default is by splitting the string by spaces into list elements, every separated word becomes a list element. From the algorithm I created thus far, we can see that the `.split()` function converts the data in variable `ip_addresses` into a list separated by spaces. It is important to note the storing of the converted list. To do this, I reassigned it back to the variable `ip_addresses` with the code: `ip_addresses = ip_addresses.split()`.

Iterate through the remove list

Now that we have the allow list of IP addresses in a list type, we can begin creating an iterative loop. By iterating through the IP addresses that are elements in the `remove_list`, we can automate the next steps to remove disallowed IP addresses. To begin the next steps, I established a `for` loop. Please see the screenshot below for reference:

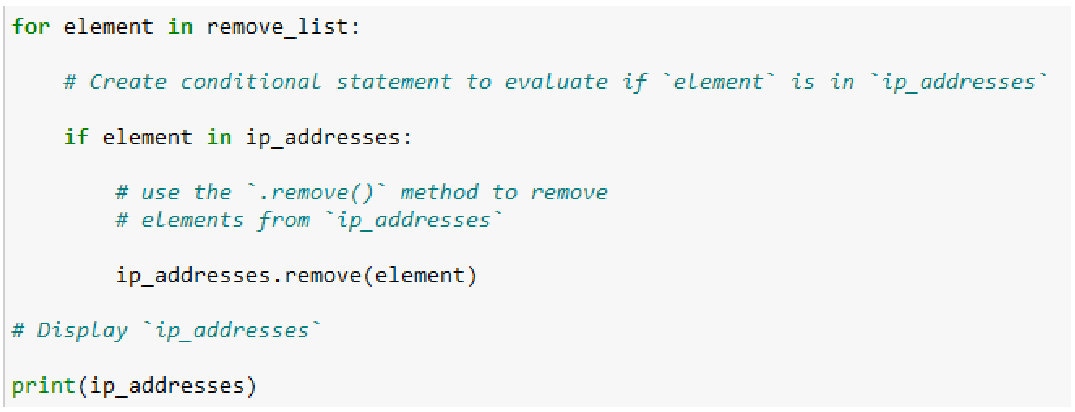


```
# Build iterative statement
# Name loop variable `element`
# Loop through `remove_list`

for element in remove_list:
```

At this stage, I want to apply a specific action to all elements in a sequence, I chose to use the `for` loop as it repeats code for a specified sequence. In this `for` loop, I named the loop variable `element` to represent each list item. Then, I used the keyword `in` to specify that I want the loop to iterate through each `element` in the `remove_list`.

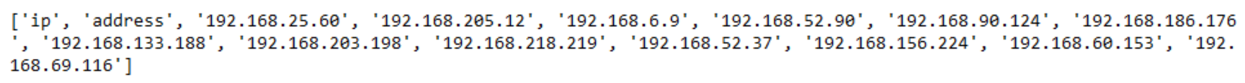
Remove IP addresses that are on the remove list

Now that the `for` loop has been created, the next step is to use a conditional statement to remove IP addresses that are on the `remove_list`. The conditional statement I used here was the `if` statement. This statement was used to compare each element in the `remove_list` to elements in the `ip_addresses`. In the body for the `if` statement I added code to the algorithm to remove any IP address from `ip_addresses` if it is found in the `remove_list`. To do this, I used the `.remove()` method with `element` as the argument. I printed the output to confirm the code is functioning properly. Please see the screenshots below for reference:



```
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)

# Display `ip_addresses`

print(ip_addresses)
```

Output of `print(ip_addresses)` after IP addresses were removed:



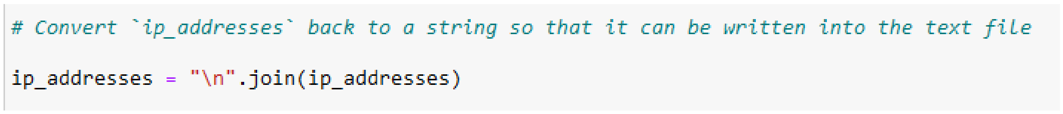
```
['ip', 'address', '192.168.25.60', '192.168.205.12', '192.168.6.9', '192.168.52.90', '192.168.90.124', '192.168.186.176
', '192.168.133.188', '192.168.203.198', '192.168.218.219', '192.168.52.37', '192.168.156.224', '192.168.60.153', '192.
168.69.116']
```

To summarize the algorithm so far, I created an automated Python script that opens the allow list, converts that string to a list named `ip_addresses`, compares `ip_addresses` variable to the `remove_list` variable, and remove any IP addresses that are identified in the `remove_list` from `ip_addresses`.

Update the file with the revised list of IP addresses

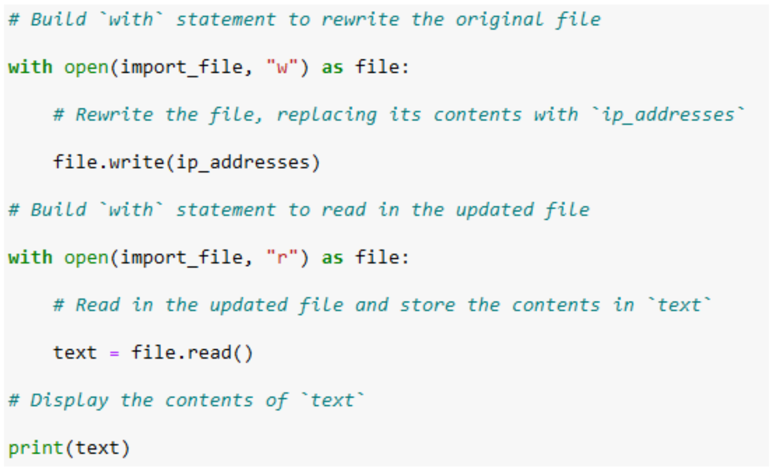
Though the algorithm seems complete, it is important to update the allow list with the new list of IP addresses so that it can be written into the text file. So far, all the algorithm did was update the allow list within the code. To fulfill this, I converted the list back into a string by using the `.join()` method.

This method can be used with a separator to tell Python where each element ends, and another begins. In my algorithm, I used the string `"\n"` as the separator to place each element on a new line. I also reassigned the new string of `ip_addresses` to `ip_addresses`. I printed the output to confirm the code is functioning properly. Please see the screenshot below for reference:



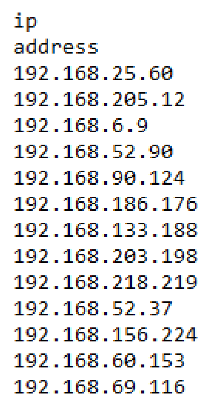
```python
# Convert `ip_addresses` back to a string so that it can be written into the text file

ip_addresses = "\n".join(ip_addresses)
```

The next step is to use another `with` statement to open the `import_file` in write mode (`“w”`) instead of opening it in read mode (`“r”`). Once the file is open and ready to be written to, I used the `.write()` method to update the `file` variable with the new `ip_addresses` variable. Please see the screenshot below for reference:



```python
# 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)
```

Output of `print(text)` after the file was updated:



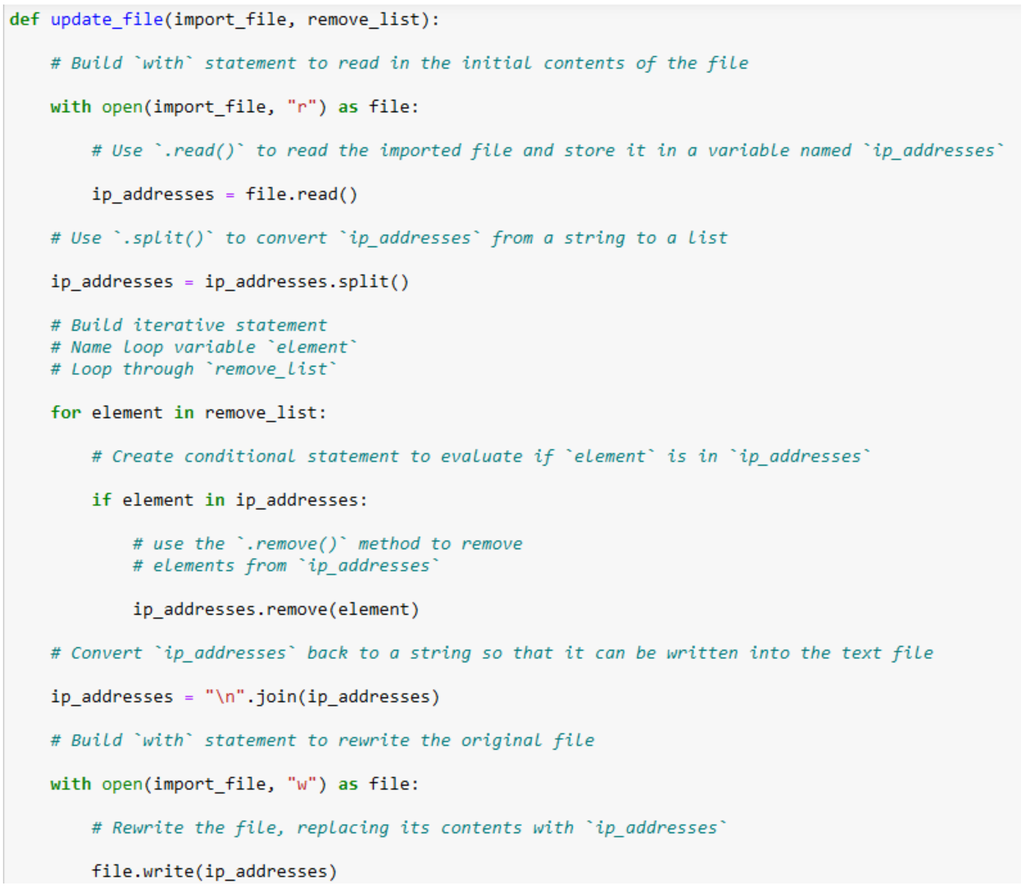
```
ip
address
192.168.25.60
192.168.205.12
192.168.6.9
192.168.52.90
192.168.90.124
192.168.186.176
192.168.133.188
192.168.203.198
192.168.218.219
192.168.52.37
192.168.156.224
192.168.60.153
192.168.69.116
```

Since I used the `with` statement and `"w"` argument in the `open()` function, I can call the `.write()` function in the body of the `with` statement. As stated earlier, this function writes string data to a

specific file, replacing any existing file content. Considering this knowledge, I was able to update the file `"allow_list.txt"` with the updated string list.
In summary, after using the `.join()` method to convert the list of `ip_addresses` into a string, I passed `ip_addresses` as an argument to the `.write()` method, overwriting the `"allow_list.txt"` with the updated IP addresses. This final step in developing my algorithm, ensures the automation of updating the `“allow_list.txt”` file by removing IP addresses that are identified in `remove_list`.

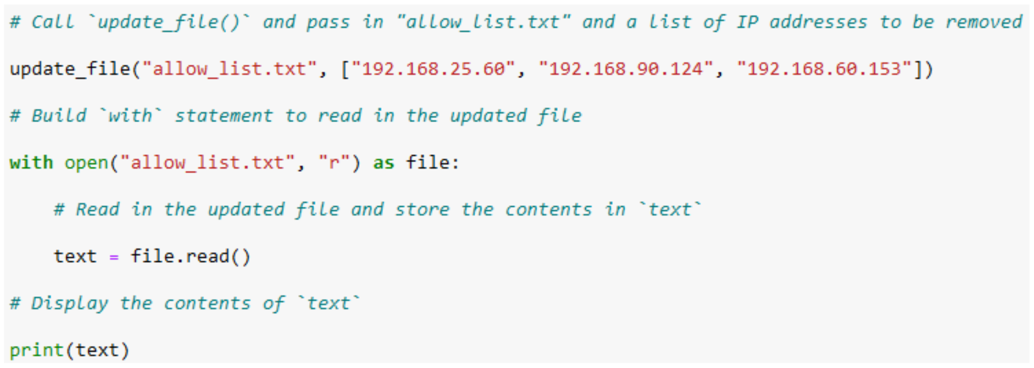
Define a function for the algorithm

Going a step further, I decided this algorithm can still be improved upon. By defining a function for the algorithm, I can make the code more organized and expand its readability. Implementing this final step, I first began with defining a function named `update_file()` that takes two parameters. The first parameter is the name of the text file that contains IP addresses, `import_file`. The second parameter is a list that contains IP addresses to be removed, `remove_list`. This will help programmers to run this code because they can simply replace the parameters with any subsequently removed IP addresses and the algorithm will still work. I printed the output to confirm the code is functioning properly. Please see the screenshot below for reference:



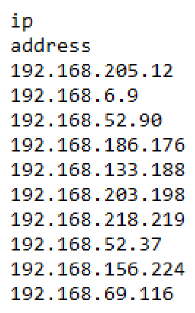
```
def update_file(import_file, remove_list):

    # 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()

    # 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)

    # 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)
```

With the algorithm now defined, I called the `update_file` function with the parameters `"allow_list.txt"` and a list of removed IP addresses. Please see the screenshot below for reference:



```
# Call `update_file()` and pass in "allow_list.txt" and a list of IP addresses to be removed

update_file("allow_list.txt", ["192.168.25.60", "192.168.90.124", "192.168.60.153"])

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

with open("allow_list.txt", "r") as file:

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

    text = file.read()

# Display the contents of `text`

print(text)
```

Output of `print(text)` after the function `update_file` was called:



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

Summary

In summary, the algorithm I created automates the removal of IP addresses that are within the `remove_list` variable from the approved IP addresses in the `"allow_list.txt"` file. My algorithm opens the allow list, converts that string to a list named `ip_addresses`, compares `ip_addresses` variable to the `remove_list` variable through iterative statements, and removes any IP addresses that are identified in the `remove_list` from `ip_addresses` by utilizing the `.remove()` method. Then, my algorithm converts the updated `ip_addresses` to a string and overwrites the contents of the `"allow_list.txt"` with the updated `ip_addresses` using the `.write()` and `join()` method. Lastly, I defined a function to the algorithm as `update_file()`. The benefits of incorporating the algorithm into a function is that it ensures the code maintains its readability and allows for further collaboration. This supports my team in executing the code, as they can substitute the parameters with any IP addresses that have been removed afterward, and the algorithm will continue to function seamlessly. Additionally, this will benefit us by increasing efficiency and productivity.