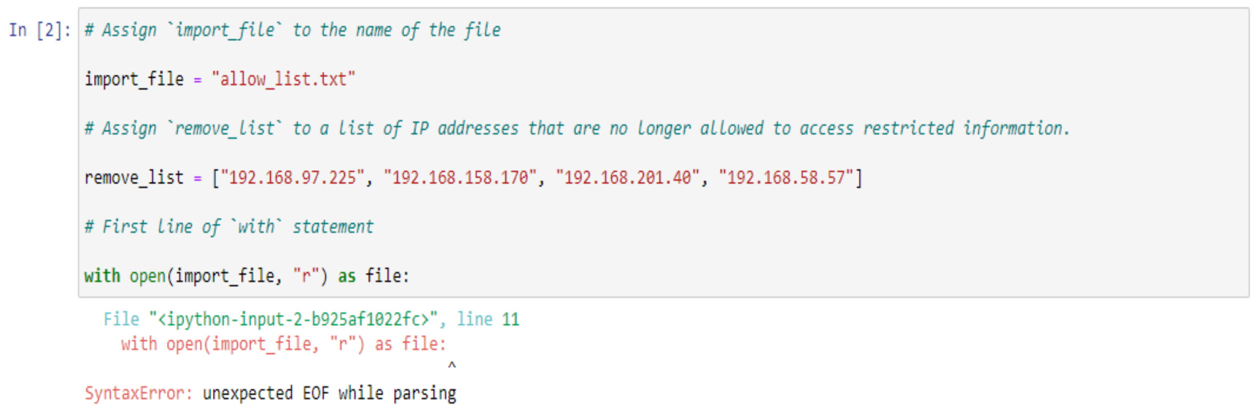
Algorithm for file updates in Python

Project description

In this scenario, I am a security professional, who's job tasks include continuously updating a file identifying employees with access to restricted content for a healthcare company. The content the file refers to is personnel patient records. Access restriction to the contents is based on employee IP addresses.

My task was to use Python code to create an algorithm that would identify any IP addresses on the "Allow" list that are present on the 'Remove" list, and then remove them from the Allow list.

Open the file that contains the allow list



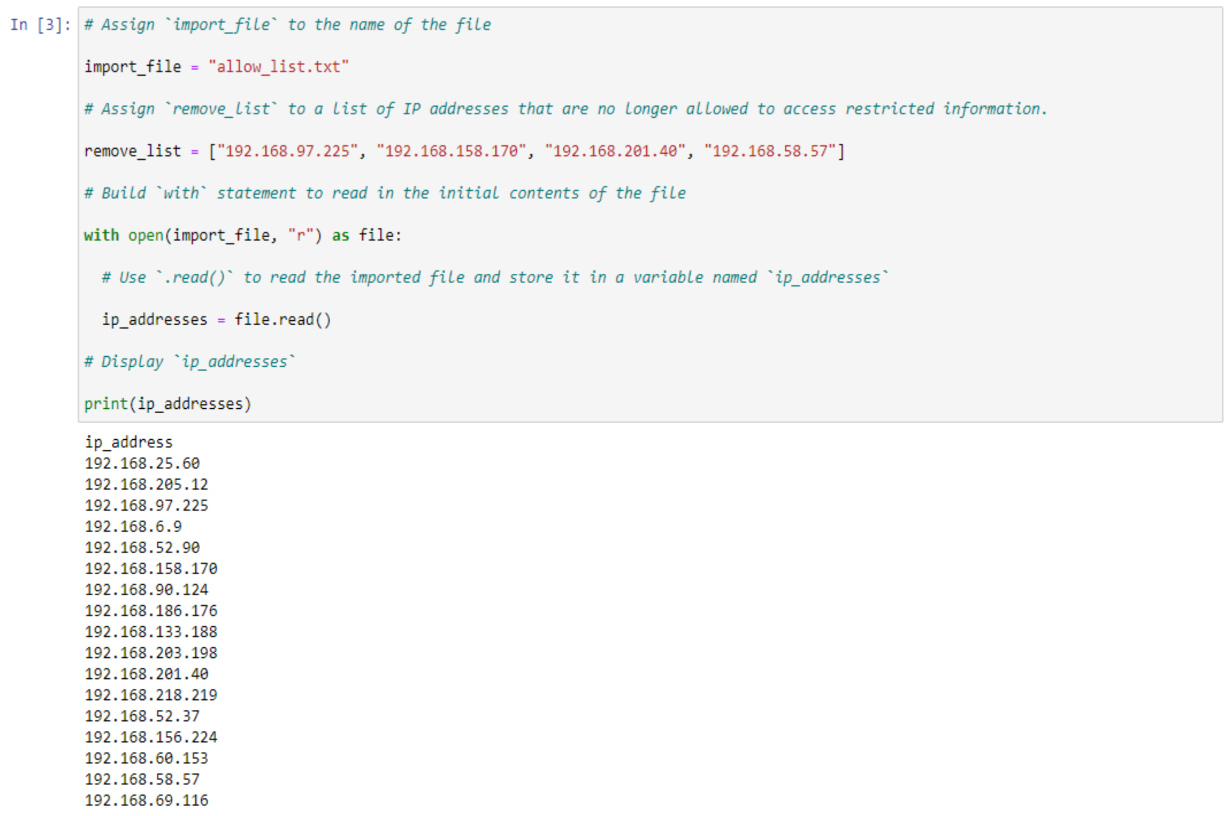
```
In [2]: # Assign `import_file` to the name of the file

        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:

          File "<ipython-input-2-b925af1022fc>", line 11
            with open(import_file, "r") as file:
                                                ^
        SyntaxError: unexpected EOF while parsing
```

The first step is to open the file containing `“allow_list.txt”` for the purpose of reading it. This file is assigned to the variable `import_file`.
I begin a `with ` statement to open it, using the file variable to store it while I work:

```
with open(import_file, “r”) as file:
```

`with` begins the statement and the `open` operator tells Python what to do. In parentheses, the first parameter dictates the txt. file through its assigned variable `import_file`, and the second parameter is the operator `“r”` (telling Python we want to read the file). `as` references the object we want to store the file in, and `file` is the object itself. An `:` is used at the end to complete the statement. Running the code at this point produces the error shown above because the `with` statement is not yet complete.

Read the file contents



```
In [3]: # 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"]

        # 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)
```

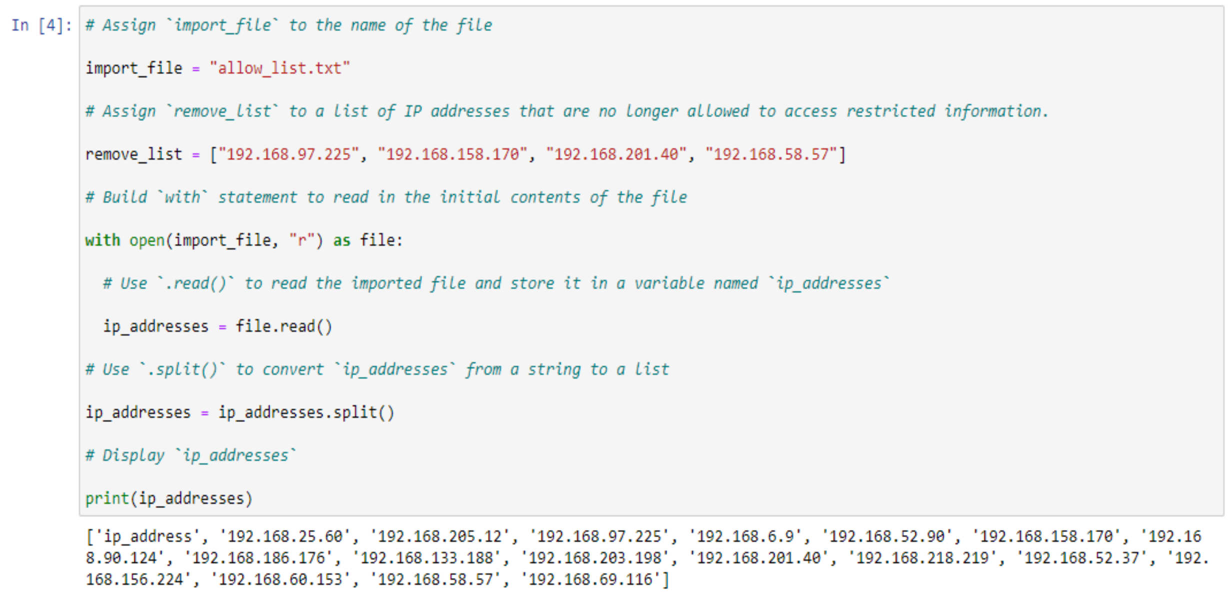
```
ip_address
192.168.25.60
192.168.205.12
192.168.97.225
192.168.6.9
192.168.52.90
192.168.158.170
192.168.90.124
192.168.186.176
192.168.133.188
192.168.203.198
192.168.201.40
192.168.218.219
192.168.52.37
192.168.156.224
192.168.60.153
192.168.58.57
192.168.69.116
```

Now that the `with` statement has been started, I assign the variable `ip_addresses` to the read function in order to print the IP addresses contained in the file:

`ip_addresses = file.read()`

Because I used `file` as my object in the with statement. I place it before the `.read()` function in order to read the imported file. The contents print as a string, one IP address per line.

Convert the string into a list



```
In [4]: # 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"]

        # 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)
```

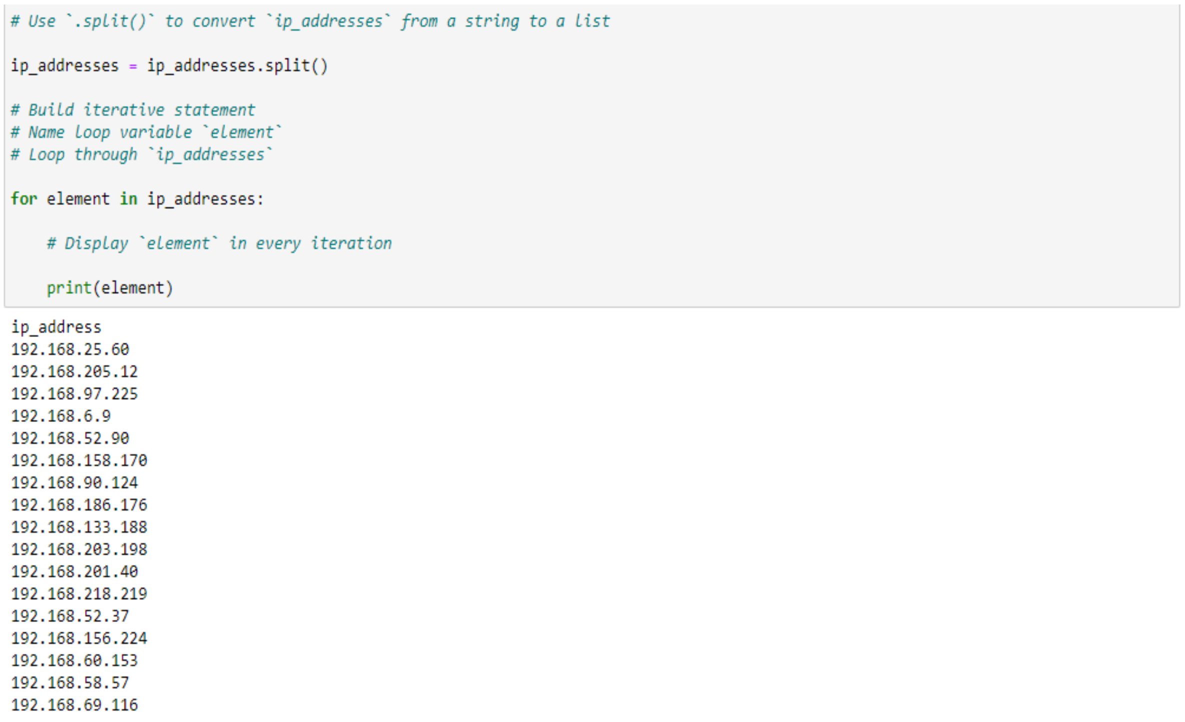
```
['ip_address', '192.168.25.60', '192.168.205.12', '192.168.97.225', '192.168.6.9', '192.168.52.90', '192.168.158.170', '192.16
8.90.124', '192.168.186.176', '192.168.133.188', '192.168.203.198', '192.168.201.40', '192.168.218.219', '192.168.52.37', '192.
168.156.224', '192.168.60.153', '192.168.58.57', '192.168.69.116']
```

The data needs to be converted from a string into list format so that I can program Python to remove individual IP's from the allow list:

`ip_addresses = ip_addresses.split()`

I place `ip_addresses` before the `.split()` function to convert the string. When I `print(ip_addesses)` now, the data outputs within brackets. By default, the `.split()` method displays the individual IP's in quotes, followed by a comma and space.

Iterate through the remove list



```
# 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 `ip_addresses`

for element in ip_addresses:

    # Display `element` in every iteration

    print(element)
```

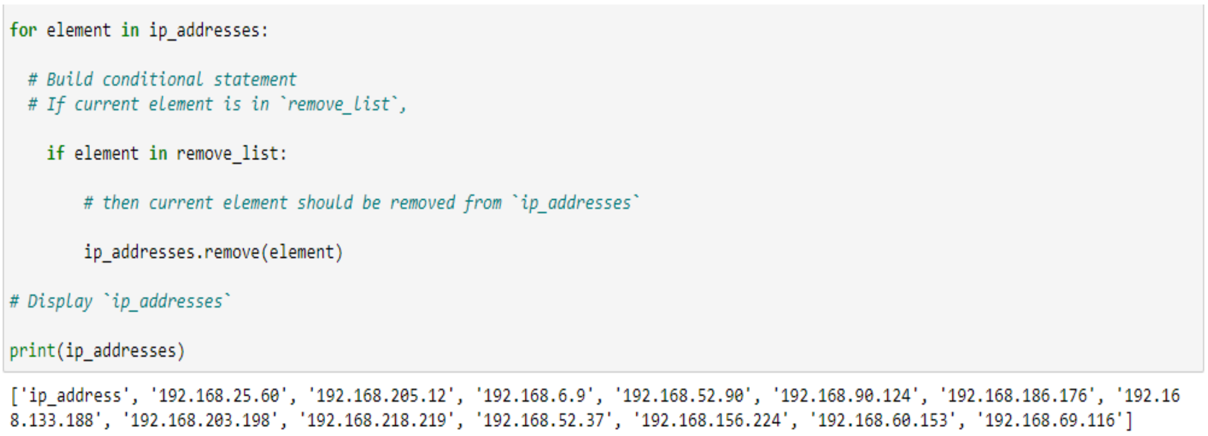
```
ip_address
192.168.25.60
192.168.205.12
192.168.97.225
192.168.6.9
192.168.52.90
192.168.158.170
192.168.90.124
192.168.186.176
192.168.133.188
192.168.203.198
192.168.201.40
192.168.218.219
192.168.52.37
192.168.156.224
192.168.60.153
192.168.58.57
192.168.69.116
```

The `remove_list` shows the IP addresses that need to be removed. I start by setting up a `for` loop that will iterate through the elements of the remove list:

`for element in ip addresses:`

`for` begins the loop. Following that, I make `element` our loop variable. `in ip_addresses` directs where the iterative statement will loop through. The `:` is needed at the end as this is the header.

Remove IP addresses that are on the remove list



```
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)
```

```
['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', '192.16
8.133.188', '192.168.203.198', '192.168.218.219', '192.168.52.37', '192.168.156.224', '192.168.60.153', '192.168.69.116']
```

Now I add code that checks if IP's in the Allow list are also in the Remove list, and takes them out if so. I start with a conditional statement:

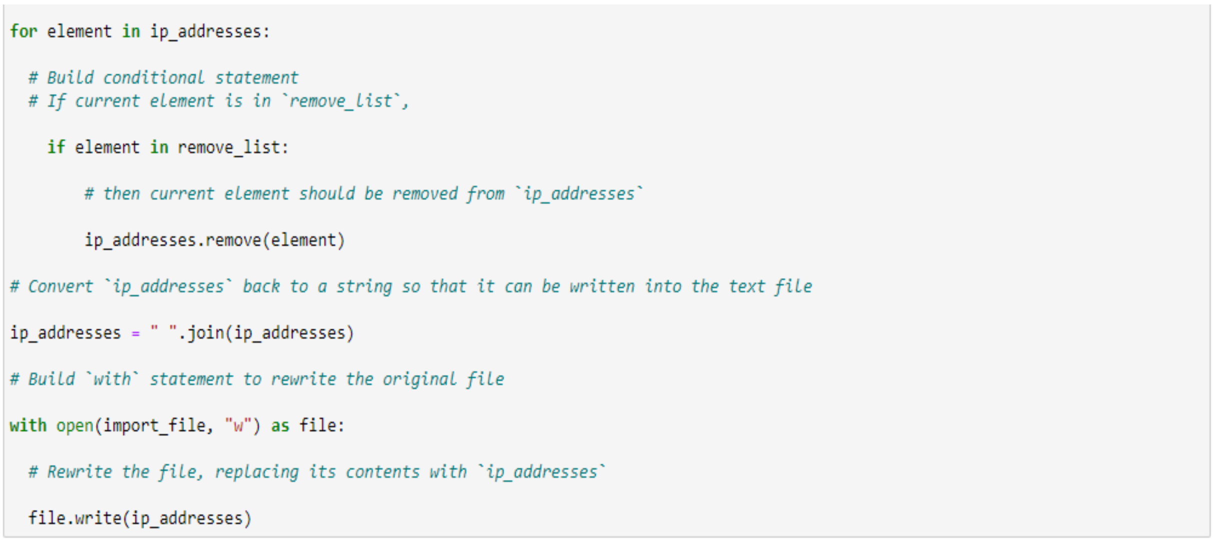
```
if element in remove_list:
```

This statement begins the body from our header, so it is indented. `if` begins the conditional, and I apply the `element` variable for the code to check. ` in remove_list` dictates where the conditional checks for the variable. This statement also needs to end with a `:`.
Now I tell Python that if the conditional is met, it will remove the current element:

```
ip_addresses.remove(element)
```

I further indent from the conditional statement above. `ip_addresses` is the variable assigned to the list we want to remove from. This is placed before the `.remove()` function. My `element` variable is placed within the parentheses so that the remove method applies to the contents in the list.

Update the file with the revised list of IP addresses



```
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)

# 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)
```

Now that the IP addresses have been removed, I complete the algorithm by updating the file with the revised list. I start with converting `ip_addresses` back into a string that can be written into the txt. file:

```
ip_ addresses = “ “.join(ip_addresses)
```

I apply the `.join()` function to our ip_addresses variable with an `=`. This method converts the list back into a string. `“ “` will apply a space in between the IP’s for readability. `ip_addresses` is placed within the parenthesis so the join applies to the data.
Finally, I use another `with` statement to rewrite the original file and call it:

```
with open(import_file, “w”) as file:

      file.write(ip_addresses)
```

For the second parameter in parenthesis, I use the `“w”` operator for Write. To call the file and rewrite it, I indent and place the object `file` from the `with` statement in front of the `.write()` function with the `ip_addresses` variable in parentheses for it to replace.

Summary

The original text file now contains an updated allow_list with the IP addresses from the `remove_list` removed. The main elements of the algorithm that make it work include the for loop, conditional statement, and .remove() method. The for loop identifies the elements of the list of IP addresses as the data I want Python to loop through. The if statement is the conditional that Python will take action on if the criteria is met (i.e. if the current element from the allow list is also in the remove list). The .remove() function is essential to the algorithm, as this tells Python the specific action to take when the conditional is met (i.e. remove the element from the allow list).