Algorithm for file updates in Python

Project description

I am a cybersecurity professional working at a private health care company. I was asked to update the list of employees that have access to sensitive information. The IP addresses of these employees are included in a file called `"allow_list.txt"`. There's also a separate list that identifies which employees must be remove from `"allow_list.txt"`. Below is an explanation of how I created an algorithm using Python to automate the update of the restricted list.

Open the file that contains the allow list

Firstly, I assigned `"allow_list.txt"` as a string belonging to variable called `import_file`:

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

import_file = "allow_list.txt"
```

As the second step, I used a `with` statement to open the file:

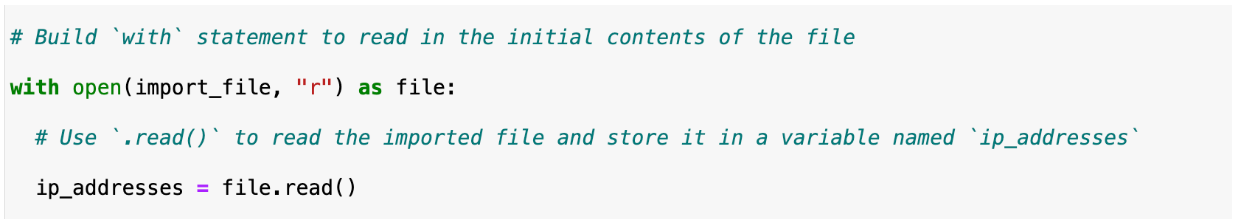
```
# First line of `with` statement

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

The `with` statement allowed me to use the `.open()` function to access the content of the file in read mode. This is possible by specifying the name of the file to open and the action I want to perform on the file, through the two parameters of the `.open()` function included within the `()`. In this case, with the first parameter I call `import_file`, whilst with the second parameter I specify that I want to read the file by including the string `“r”`. The code also uses the `as` keyword to assign a variable named `file` which stores the output of the `.open()` function. `with` automatically closes the file after the `with` statement has run.

Read the file contents

To be able to read its content, the file has to be converted into a string. For this purpose, I used the `.read()` method:



```
# 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()
```

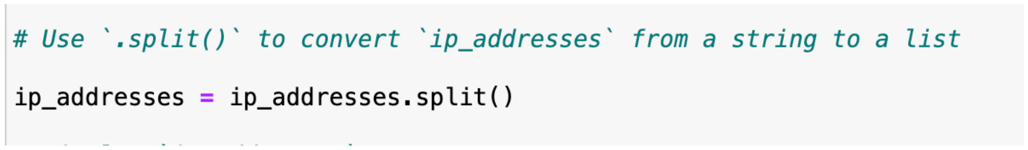
I can now call a `.read()` method directly within the body of `with`. `.read()` directly converts files into strings that are readable in Python. For that reason, I used the following syntax:

`ip_addresses = file.read()`

This allowed me to convert the content of `file` into a string. At the same time, I assigned it to a new variable called `ip_addresses`, to simplify usability in the future.

Convert the string into a list

To remove individual IP addresses from the `"allow_list.txt"`, I needed its content to be in list format. Therefore, I next used the `.split()` method to convert the `ip_addresses` string into a list:



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

The `.split()` function is called by appending it to a string variable. The purpose of splitting `ip_addresses` into a list is to make it easier to remove IP addresses from the allow list. By default, the `.split()` function splits the text by whitespace into list elements. In this algorithm, the `.split()` function takes the data stored in the variable `ip_addresses`, a string of elements each separated by a whitespace, and converts it into a list. To store this list, I reassigned it back to the variable `ip_addresses`.

Iterate through the remove list

At this point, I needed to iterate through the list of IP addresses within `ip_addresses.` To do this, I used a `for` loop, which, in Python, repeats code for a specific sequence.

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

for element in ip_addresses:
```

The overall purpose of the `for` loop in a Python algorithm like this 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`.

Remove IP addresses that are on the remove list

Moving on, I needed to build a conditional statement that would allow me to remove the elements of `ip_addresses` that are also contained in `remove_list`. I placed the statement inside the body of the `for` loop:

```python
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)
```

The conditional statement is opened by the `if` keyword. In essence, the statement is saying that if any `element` of the `ip_addresses` list is also found in `remove_list` list, then the specific `element` should be removed from `ip_addresses`. I achieved this by appending the `.remove()` method using the following syntax:

`ip_addresses.remove(element)`

`element` was assigned as the argument of the `.remove()` method. This way the loop iteration was able to remove each IP addresses marked in the `remove_list` from the `ip_addresses` list.

Update the file with the revised list of IP addresses

Next step was about updating the original allow list file with the revised list of authorized IP addresses. The first thing I needed to do was to convert the list back into a string using 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()` method takes in an iterable as its argument and concatenates every element of it into a string. In this case, the argument of the `.join()` method is `ip_addresses`. The `.join()` method is appended to a string consisting of the character that will be used to separate every element in the iterable once this is converted into a string. In my case, I use the string `"\n"`, which instructs Python to place each element into a new line. I reassigned the syntax to the same variable `ip_addresses`, so I was able to use it later in conjunction with the `.write()` method.

```
# 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)
```

In the following step, the write statement allowed me to open the original `import_file` using `.open()`, once again. In this case, the second argument of the `.open()` method was `”w”` which clarifies that I want to overwrite the content already present in `import_file`.

To make sure that the data in `"allow_list.txt"` is updated and that only authorized users can access restricted information, I needed to write over the existing content of the allow list using the `.write()` method. I did this by appending `.write()` to the `file` object and passing the `ip_addresses` variable as its argument.

Project 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 `ip_addresses`. With each iteration, I evaluated if the element was part of the `remove_list` 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.