Algorithm for file updates in Python

Project description

While working at a health care company, I am tasked with regularly updating a file that identifies the employees who can access restricted content.

Using Python, I will develop an algorithm to check whether the allow list contains any IP addresses identified on the remove list. If so, I will remove those IP addresses from the file containing the allow list.

Open the file that contains the allow list

In order to open the "allow\_list.txt", I need to first assign a variable to this file, which I did assigning import\_file to it. Then, I need to use a with statement with the open() function. When using with, Python knows to automatically release resources that would otherwise keep our system busy until the program finishes running. The open() function opens a file, with the two arguments (import\_file, "r") used to define what file we are opening and how it is being used. import\_file is the variable of the .txt file we wish to open, and "r" tells Python we want to read the file.

After the argument (import\_file, "r"), as file: contains the file information in the with statement, and end with a colon (:)



Read the file contents

In order to read the file contents, we have to assign the .read() method to a variable I will name ip\_addresses. Since we called the with statement as file, we then use this with .read(). When put together, looks like ip\_addresses = file.read().This will convert the ip\_addresses into a string. We finally use a print() function to print out the ip\_addresses string to test to see if everything so far is correct.



Notice that the string has one element per line. This will be useful information to know in the next step.

Convert the string into a list

Since a string is immutable, we will need to convert the IP addresses to a list format. In order to do this, we will need to use the .split() method to convert the ip\_addresses string into a list. Because the .split() method converts a string to a list, and we need to do this for the ip\_addresses string, the syntax looks like this: ip\_addresses = ip\_addresses.split().

When we printed out the string in the previous step, each element of the string was separated by whitespace. Because of this, adding no arguments to the .split() function defaults to splitting the string on whitespace. This will work well for us since each IP address is on a different line.



Notice that the output that used to be a string is now in the form of a list.

Iterate through the remove list

The first screenshot showed a second list called remove\_list, which contains all of the IP addresses that should be removed from the ip\_addresses list. In order to check each IP address against the remove\_list, we first need to iterate through this list. We will do this with a for loop with element as the loop variable: for element in ip\_addresses:

This is the setup we are going to use to begin the iteration. for starts the loop and in is the loop condition, which is looping through ip\_addresses. We always make sure to end on a colon (:)

Again, we then use the print() function to debug the code if necessary.



Remove IP addresses that are on the remove list

Now that we have made the for loop, and started iterating through the remove\_list, we need to create a conditional that evaluates if the loop variable element is part of the ip\_addresses list. Then, inside that conditional, we use the .remove() method to the ip\_addresses list and remove the IP addresses that were evaluated as True.

While inside the for loop, we need to create an if statement. if the current element is in remove\_list: if element == remove\_list: if this evaluates to True, then we move to the next line of the statement, ip\_addresses.remove(element). This tells us to remove the element from the ip\_addresses list.

Ensuring proper indentation is key here, as everything is inside the for loop, but the second line of the if statement should only be run inside this statement.



Update the file with the revised list of IP addresses

Now that we have removed the IP addresses, I can complete the algorithm by updating the file with this revised list. To do this, we must convert the ip\_addresses back into a string using the .join() method.

In order to convert a list back to a string, we need to use the .join() method. This method will concatenate the element of a list into a string. We must be careful when using .join() because the syntax of this method is different than many common ones.

The syntax of .join() starts off with how we will separate the elements. Recall back to the .split() function: we left the argument as a default to separate the elements. This does not work with .join()

ip\_addresses = "\n".join(ip\_addresses). This tells us that the list ip\_addresses is now going back into a string. "\n" is the input to have every element on its own line.

Finaly, I use a with statement again to write over the original file. The syntax is the same as the last with statement we did, however, instead of "r" to read, we use "w" to write over the file. We then finish the statement with file.write(ip\_address) which tells us that Python will be rewriting the ip\_addresses string to the file.



Summary

I created an algorithm to determine if an IP address was on the blocked IP address list. We had to open, read, and write over files. Using algorithms such as this will allow a cybersecurity professional to minimize human error and maximize efficiency.