# Algorithm for file updates in Python

## Project description

At our company, we manage access to restricted content by maintaining an allow list of IP addresses in the "allow\_list.txt" file. Additionally, there is a remove list that specifies the IP addresses that should be denied access. I developed an algorithm to automate the process of updating the "allow\_list.txt" file by removing the IP addresses listed in the remove list.

## Open the file that contains the allow list

For the first step of the algorithm, I opened the "allow\_list.txt" file by assigning the file name to a variable named import\_file like this:



Then, I used a with statement to open the file:



In my algorithm, I use the with statement along with the open() function in read mode to open the "allow\_list.txt" file. This approach allows me to access the IP addresses stored in the file. The with keyword is beneficial as it manages the file resources efficiently by automatically closing the file once the operations within the block are completed. The syntax with open(import\_file, "r") as file: uses open() with two parameters: the first specifies the file to open, and the second, "r", indicates that the file is to be opened in read mode. Inside the with block, the as keyword assigns the open file to a variable named file, which holds the file's content while the with statement is in effect.

## 

## Read the file contents

To read the contents of the file, I used the .read() method, which converts the file's contents into a string.



When I open a file using the open() function with the "r" argument for "read," I can then use the .read() method within the with statement's body. This method transforms the file's content into a string, making it readable. I apply the .read() method to the file variable specified in the with statement. The result, a string containing the file's contents, is then stored in a variable named ip\_addresses.

In short , this code snippet converts the "allow\_list.txt" file into a string format, setting the stage for further data manipulation and extraction in my Python program.

## Convert the string into a list

To facilitate the removal of individual IP addresses from the allow list, it was necessary to convert the data into a list format. For this purpose, I used the .split() method to transform the ip\_addresses string into a list. This method breaks the string into separate elements based on a delimiter, which by default is any whitespace, including new lines and spaces, thus creating a list of IP addresses:



The .split() function is utilized by appending it to the string variable containing the IP addresses. It operates by splitting the string into a list based on whitespace, which is the default delimiter. In this algorithm, the main goal of splitting ip\_addresses into a list is to facilitate the process of removing specific IP addresses from the allow list. This function takes the data from the ip\_addresses variable—a string where IP addresses are separated by whitespace—and converts it into a list of individual IP addresses. After splitting, I reassigned this list back to the same variable, ip\_addresses, to update it with the list format.

## Iterate through the remove list

A crucial step in my algorithm is to iterate over the IP addresses that are elements in the remove\_list. To accomplish this, I included a for loop in the code. This loop goes through each IP address in the remove\_list and performs actions on them, such as checking and removing these addresses from the ip\_addresses list if they are found. The for loop is an efficient way to handle each element individually and modify the ip\_addresses list accordingly.



In Python, the for loop is used to execute a block of code repeatedly for each item in a specified sequence. In the context of this algorithm, the purpose of the for loop is to systematically apply a set of operations to every element in a sequence. The loop starts with the for keyword, followed by a loop variable, element, and the in keyword. The in keyword specifies that the loop should iterate through the sequence ip\_addresses, assigning each value sequentially to the loop variable element. This structure allows each IP address in the sequence to be processed individually according to the code within the loop.

## Remove IP addresses that are on the remove list

To remove any IP address from the ip\_addresses list that also appears in the remove\_list, I used a straightforward method, especially since there were no duplicates in the ip\_addresses. This simplifies the removal process because each IP address only needs to be checked once. The code I used for this is:



Within the for loop in my algorithm, I first set up a conditional check to determine if the loop variable, element, was present in the ip\_addresses list. This step was crucial because attempting to remove an element that does not exist in the list using the .remove() method would lead to an error.

In this code:

* The for loop iterates through each IP address stored in remove\_list.
* The if statement checks if the current element is found within ip\_addresses.
* If the condition is true, the .remove() method is called on ip\_addresses, with the element passed as an argument, thereby removing the specified IP address from the ip\_addresses list if it exists. This ensures that each IP address in the remove\_list is effectively removed from ip\_addresses.

## Update the file with the revised list of IP addresses

As the final step in my algorithm, I needed to update the "allow\_list.txt" file with the revised list of IP addresses. To accomplish this, it was necessary to convert the list of IP addresses back into a single string format. I achieved this using the .join() method, which efficiently concatenates all elements of the list into a single string, each separated by a newline character. Here’s how I implemented it:



In this line of code:

* "\n".join(ip\_addresses) takes each IP address from the ip\_addresses list and joins them into a single string, where each IP address is separated by a newline (\n). This format is suitable for writing back to the file as it ensures each IP address is on a new line, matching the original file format.

After converting the list back into a string, I then proceeded to open the "allow\_list.txt" file in write mode and overwrite it with the updated list:



This block of code opens the allow list file in write mode ("w"), which allows for writing the new, updated string of IP addresses back to the file, effectively updating it with the revised list.

## Summary

I developed an algorithm designed to eliminate IP addresses from the "allow\_list.txt" file, which holds a list of approved IPs. The steps of the algorithm are as follows:

1. **Open the File:** I began by opening the "allow\_list.txt" file and reading its contents, which I then converted into a string.
2. **Convert to List:** Next, I transformed this string into a list of IP addresses, stored in the variable ip\_addresses.
3. **Iterate and Remove:** I iterated over each IP address in the remove\_list. During each iteration, I checked if the current address was present in the ip\_addresses list. If found, I used the .remove() method to delete the address from ip\_addresses.
4. **Convert Back to String:** After removing the specified IPs, I converted the updated list of IP addresses back into a single string using the .join() method.
5. **Write to File:** Finally, I opened the "allow\_list.txt" again in write mode to overwrite it with the newly revised list of IP addresses.

This algorithm effectively updates the "allow\_list.txt" file, ensuring that any IP addresses specified in the remove\_list are no longer allowed access.