# Algorithm for file updates in Python

## Project description

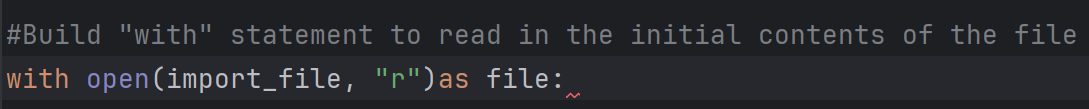
At my organization, access to restricted content is controlled with an allow list of IP addresses. The "allow\_list.txt" file identifies these IP addresses. A separate remove list identifies IP addresses that should no longer have access to this content. I created an algorithm to automate updating the "allow\_list.txt" file and remove these IP addresses that should no longer have access.

## Open the file that contains the allow list

For the first part of the algorithm, I opened the "allow\_list.txt" file. First, I assigned this file name as a string to the import\_file variable:

## 

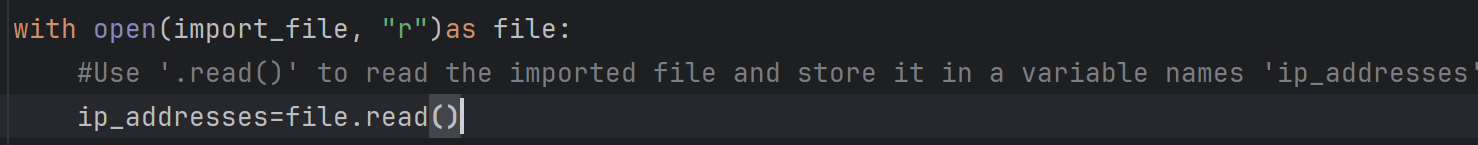
Then, I used a with statement to open the file:



My algorithm employs a with statement alongside the .open() function in read mode ("r") to access the contents of the allow list file. This action is necessary to retrieve the stored IP addresses. The with keyword ensures efficient resource management by automatically closing the allow list file once the reading operation is complete. Specifically, the line with open(import\_file, "r") as file: first uses .open() with two arguments: the file path (import\_file) and the read mode indicator ("r"). The as keyword then assigns the opened file object to the variable file, which is used to interact with the file's data within the with block.

## Read the file contents

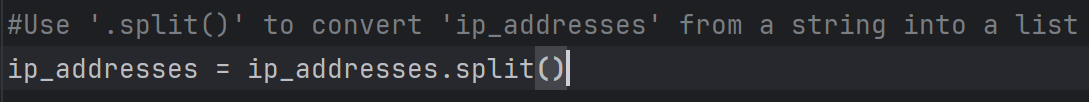
In order to read the file contents, I used the .read() method to convert it into the string.



Using the .open() function with the "r" argument for reading enables the use of the .read() function within the with block. The .read() method transforms the entire file content into a single string, making it readable within the program. In my code, I applied .read() to the file variable (assigned during the with statement) and stored the resulting string in the ip\_addresses variable. Essentially, this part of the code reads the entire "allow\_list.txt" file and converts its content into a string format, which I can then process to organize and extract the IP address data in my Python program.

## Convert the string into a list

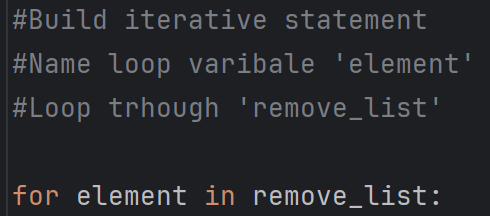
In order to remove individual IP addresses from the allow list, I needed it to be in list format. Therefore, I next used the .split() method to convert the ip\_addresses string into a list:



The .split() function, when added to a string variable, transforms the string's content into a list. I'm using .split() on the ip\_addresses string to create a list, which will simplify the process of removing IP addresses from the allow list later in the algorithm. By default, .split() separates the string into list items based on whitespace. In this case, it takes the ip\_addresses string, where each IP is separated by spaces, and converts it into a list of individual IP addresses. This resulting list is then stored back in the ip\_addresses variable, overwriting the original string.

## Iterate through the remove list

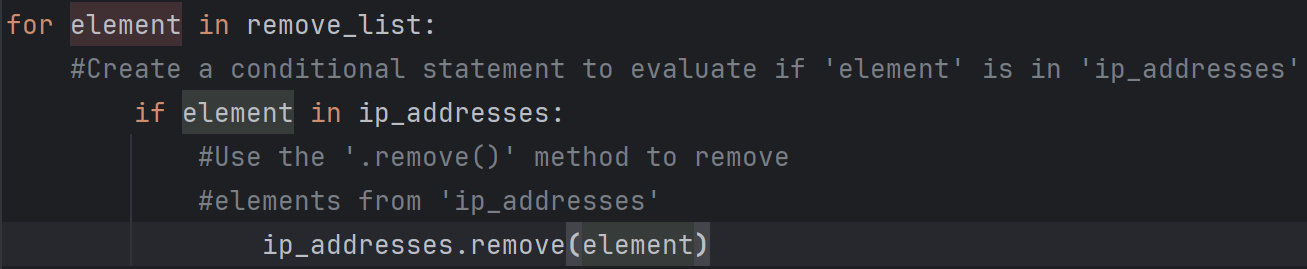
A key part of my algorithm involves iterating through the IP addresses that are elements in the remove\_list. To do this, I incorporated a for loop:



In Python, a for loop executes a block of code repeatedly for each item within a given sequence. In this algorithm, the for loop's role is to apply a set of instructions to every individual IP address within the ip\_addresses sequence (which is now a list). The loop begins with the for keyword, followed by element (the variable that will temporarily hold each IP address during each iteration), and the in keyword. The in keyword signifies that the loop will iterate through the ip\_addresses list, assigning one IP address at a time to the element variable for processing within the loop's body.

## Remove IP addresses that are on the remove list

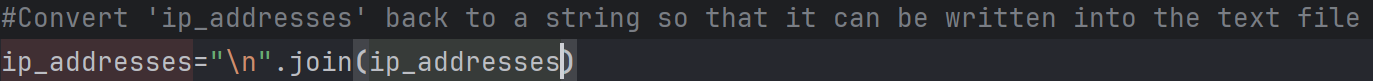
My algorithm requires removing any IP address from the allow list, ip\_addresses, that is also contained in remove\_list. Because there were not any duplicates in ip\_addresses, I was able to use the following code to do this:



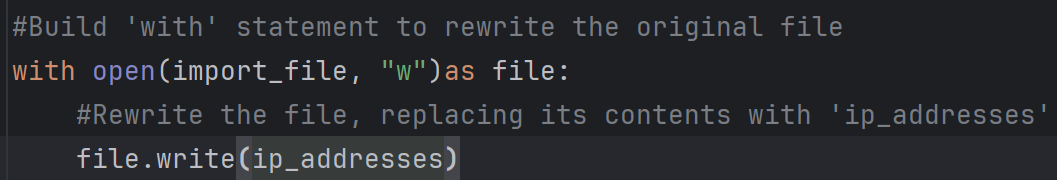
Inside the for loop, I first implemented a check to see if the current element (representing an IP address from the remove\_list) actually exists within the ip\_addresses list. This check is crucial because attempting to use the .remove() method on an element that isn't present in the list would cause the program to crash with an error. Only if the IP address from the remove\_list is found within ip\_addresses does the code proceed to the next step. Within this if condition, I then used the .remove() method on the ip\_addresses list, passing the current element as the argument. This ensures that for each IP address present in the remove\_list, the corresponding IP address is removed from the ip\_addresses list.

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

As a final step in my algorithm, I needed to update the allow list file with the revised list of IP addresses. To do so, I first needed to convert the list back into a string. I used the .join() method for this:



The .join() method takes all the items within a list (or any iterable) and concatenates them into a single string. This method is called on a string that defines the separator to be used between the joined elements. In this algorithm, I employed .join() to convert the ip\_addresses list back into a string format, specifically so that it could be written to the "allow\_list.txt" file using the .write() method. I used the newline character ("\n") as the separator to ensure that each IP address in the resulting string would be placed on a separate line in the file. Following this, I used another with statement along with the .write() method to actually update the content of the "allow\_list.txt" file with this newly formatted string of IP addresses.



This time, I opened the "allow\_list.txt" file in write mode ("w") using the open() function within a with statement. The "w" argument signifies that I intend to write data to the file, and importantly, it will overwrite any content that already exists in the file. When a file is opened in write mode, the .write() function becomes available within the with block. The .write() function takes a string as input and writes it to the specified file, completely replacing any previous content. In this scenario, my goal was to update the "allow\_list.txt" file with the modified list of IP addresses (now in string format). To achieve this, I called the .write() function on the file object (which represents the opened file) and passed the ip\_addresses variable as the argument. This action effectively replaces the entire content of "allow\_list.txt" with the updated string of allowed IP addresses, thus restricting access for any IPs that were removed.

## Summary

This algorithm I developed updates the "allow\_list.txt" file by removing specific IP addresses listed in the remove\_list variable. The process begins by opening the "allow\_list.txt" file, reading its content into a string, and then transforming that string into a list, which is stored in the ip\_addresses variable. Next, the algorithm iterates through each IP address in the remove\_list. For each IP, it checks if that IP exists in the ip\_addresses list. If a match is found, the .remove() method is used to delete that IP from the ip\_addresses list. Finally, the modified ip\_addresses list is converted back into a single string using the .join() method, and this string then overwrites the original content of the "allow\_list.txt" file, effectively updating the list of approved IP addresses.