# Algorithm for file updates in Python

## Project description

In my workplace, we manage access to restricted content by using an IP address allow list. We use a file called "allow\_list.txt" to specify the authorized IP addresses. Additionally, there's a separate list known as the remove list, which contains IP addresses that are no longer allowed to access this content. To streamline this process, I developed an algorithm that automatically updates the "allow\_list.txt" file and removes the IP addresses that should no longer have access.

## Open the file that contains the allow list

To start the algorithm, I began by opening the "allow\_list.txt" file. Initially, I stored the name of this file as a string in a variable called import\_file.



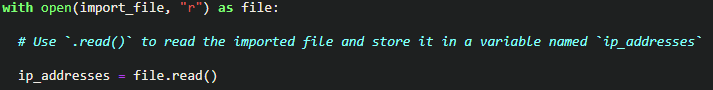
Then, I used a ‘with’ statement to open the file:



In my algorithm, I employ the 'with' statement along with the '.open()' function set to read mode to initiate the opening of the allow list file for reading purposes. The primary objective here is to gain access to the IP addresses contained within the allow list file. The use of the 'with' keyword is crucial because it assists in resource management by automatically closing the file upon exiting the 'with' statement. In the code snippet 'with open(import\_file, "r") as file:', you can observe the 'open()' function with two parameters. The first parameter specifies the file to be imported, and the second one indicates the desired operation on the file. In this scenario, "r" signifies that I intend to read the file. Additionally, the 'as' keyword is used to assign the variable 'file', which stores the output of the '.open()' function while I perform actions within the 'with' statement.

## Read the file contents

To access the content of the file, I utilized the '.read()' method, which transformed the file's content into a string.



When using the '.open()' function with the "r" argument for "read," I can invoke the '.read()' function within the 'with' statement's body. This '.read()' method is responsible for converting the file into a string, enabling me to read its contents. I applied this '.read()' method to the 'file' variable specified within the 'with' statement. Subsequently, I assigned the resulting string to a variable called 'ip\_addresses.'

In essence, this piece of code accomplishes the task of reading the contents of the "allow\_list.txt" file and converting them into a string format. This string can then be utilized later in my Python program for data organization and extraction.

## Convert the string into a list

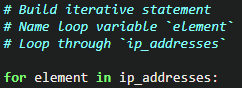
To enable the removal of individual IP addresses from the allow list, it was necessary to have the data in list format. Consequently, I proceeded to utilize the '.split()' method, which allowed me to transform the 'ip\_addresses' string into a list format.



The '.split()' function is invoked by adding it to a string variable, and its function is to convert the contents of a string into a list. The objective behind splitting 'ip\_addresses' into a list is to facilitate the removal of IP addresses from the allow list. By default, the '.split()' function divides the text into list elements wherever there is whitespace. In this algorithm, the '.split()' function takes the data stored in the 'ip\_addresses' variable, which contains a string of IP addresses separated by whitespace, and transforms this string into a list of individual IP addresses. To preserve this list, I reassigned it to the 'ip\_addresses' variable.

## Iterate through the remove list

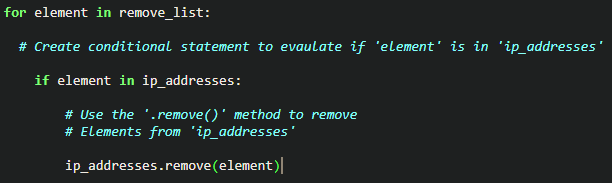
An essential component of my algorithm revolves around the process of looping through the IP addresses listed in the 'remove\_list.' To achieve this, I integrated a 'for' loop.



In Python, the 'for' loop is used to execute code repeatedly for a defined sequence. In a Python algorithm like this one, the primary objective of the 'for' loop is to execute particular code statements for every element within a sequence. The 'for' keyword initiates the 'for' loop, followed by the loop variable 'element,' and then the 'in' keyword. The 'in' keyword signifies the iteration process through the 'ip\_addresses' sequence and assigns each value to the loop variable 'element.'

## Remove IP addresses that are on the remove list

In my algorithm, the task involves eliminating any IP address from the 'ip\_addresses' allow list if it also appears in the 'remove\_list.' Since there were no duplicates in the 'ip\_addresses' list, I employed the following code to accomplish this.



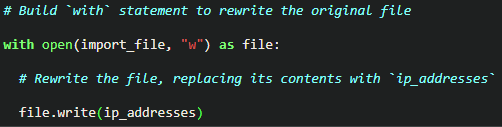
Initially, inside my 'for' loop, I established a condition to assess whether the loop variable 'element' was present in the 'ip\_addresses' list. This step was necessary to prevent errors, as using the '.remove()' method on elements not existing in 'ip\_addresses' would lead to issues. Subsequently, within that condition, I utilized the '.remove()' method on 'ip\_addresses.' I supplied the loop variable 'element' as the argument, ensuring that every IP address from the 'remove\_list' would be successfully removed from 'ip\_addresses.'

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

In the last phase of my algorithm, the task was to refresh the allow list file with the updated set of IP addresses. To achieve this, my initial requirement was to transform the list back into a string. To accomplish this transformation, I employed the '.join()' method.



The '.join()' method combines all elements within an iterable and forms them into a single string. This method is applied to a string that contains characters specifying how the elements within the iterable should be separated when they are joined into a string. In this algorithm, I utilized the '.join()' method to convert the 'ip\_addresses' list into a string. This string was then passed as an argument to the '.write()' method when updating the "allow\_list.txt" file. I designated the string ("\n") as the separator, directing Python to place each element on a new line. Following this, I employed another 'with' statement along with the '.write()' method to execute the file update.



On this occasion, within my 'with' statement, I introduced a second argument, "w," when using the 'open()' function. This specific argument signifies my intention to open a file for the purpose of writing and overwriting its existing content. With the "w" argument, I have the capability to invoke the '.write()' function within the 'with' statement's body. The '.write()' function is responsible for writing string data into a designated file, effectively replacing any previous content. In this particular scenario, I aimed to write the updated allow list as a string back to the "allow\_list.txt" file. By doing so, I ensure that the restricted content becomes inaccessible to any IP addresses that were removed from the allow list. To carry out this file rewrite operation, I appended the '.write()' function to the 'file' object, which I previously identified in the 'with' statement. I provided the 'ip\_addresses' variable as the argument, specifying that the contents of the file specified in the 'with' statement should be replaced with the data from this variable.

## Summary

I developed an algorithm designed to eliminate IP addresses found in a 'remove\_list' variable from the "allow\_list.txt" file, which contains a list of approved IP addresses. This algorithm comprised several steps. First, I opened the file, then transformed its content into a string for reading purposes. Subsequently, I converted this string into a list stored in a variable named 'ip\_addresses.'

Next, I initiated an iteration process through the IP addresses in the 'remove\_list.' During each iteration, I checked whether the element existed within the 'ip\_addresses' list. If it was present, I applied the '.remove()' method to eliminate the element from 'ip\_addresses.' Following this, I utilized the '.join()' method to convert 'ip\_addresses' back into a string, facilitating the task of overwriting the contents of the "allow\_list.txt" file with the updated list of IP addresses.