# Algorithm for file updates in Python

## Project description

Within my organization, access to restricted content is managed through an allow list of IP addresses. The file "allow\_list.txt" contains these authorized IP addresses. Additionally, there's a separate removal list, which identifies IP addresses that should be revoked access. I've devised an algorithm to automate the updating of the "allow\_list.txt" file, removing these unauthorized IP addresses from accessing the content.

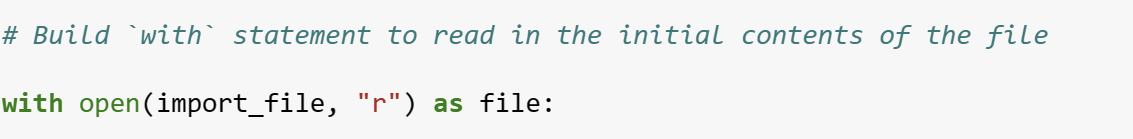
## Open the file that contains the allow list

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:

## 

I then used the **“with”** statement to open up the file:



In my algorithm, I utilize the **with** statement alongside the **.open()** function set to read mode to access the allow list file. This enables me to retrieve the IP addresses stored within. The **with** keyword manages resources effectively by automatically closing the file once the **with** statement is exited. The code snippet **with open(import\_file, "r")** **as file:** contains two parameters within the **open()** function. The first parameter specifies the file to be imported, while the second indicates the intended action, which in this case is reading **("r")**. Furthermore, the **as** keyword assigns the variable named **file,** which holds the output of the **.open()** function for use within the **with** statement.

## Read the file contents

To read the contents of a file I used the **.read()** method to convert it into the string.



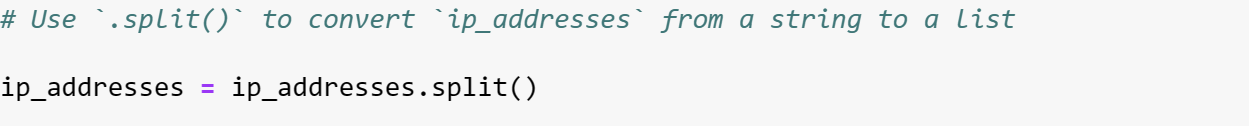
When utilizing an **.open()** function with the **"r"** argument for reading, I can invoke the **.read()** method within the with statement block. This method converts the file content into a string, enabling me to access its contents. I've applied the **.read()** method to the file variable defined in the with statement, storing its string output in the variable **ip\_addresses.**

To recap, this code extracts the contents of the **"allow\_list.txt"** file as a string, facilitating subsequent organization and data extraction within my Python program.

## Convert the string into a list

To remove individual IP addresses from the allow list, I needed it to be in list format.

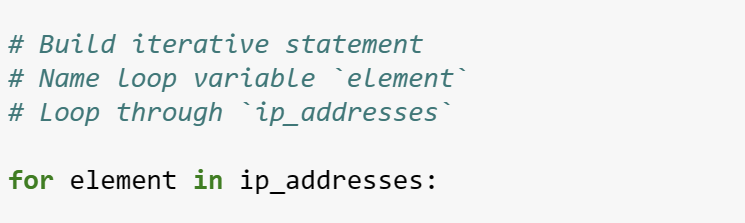
So i used the **.split()** method to convert the **ip\_addresses** string into a list:



To utilize the **.split()** functionality, it's appended to a string variable. This method operates by transforming the string's contents into a list. The objective of splitting **ip\_addresses** into a list is to simplify the removal of IP addresses from the allow list. By default, **.split()** divides the text by whitespace into list elements. In this algorithm, **.split()** processes the data contained in the **ip\_addresses** variable, which consists of a string of IP addresses separated by whitespace, converting it into a list. To preserve this list, I reassigned it to the **ip\_addresses** variable.

## Iterate through the remove list

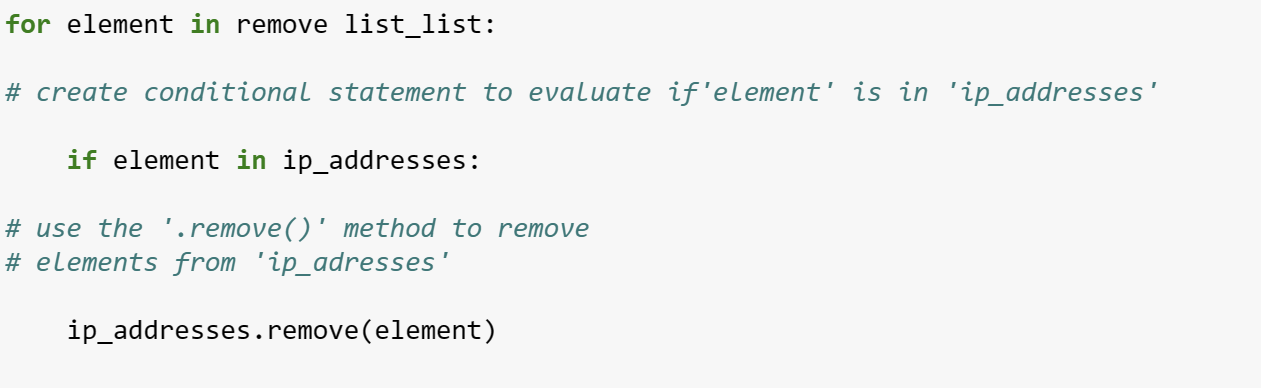
Central to my algorithm's functionality is the iteration through the IP addresses listed in the **remove\_list**. To achieve this, I integrated a **for** loop:



In Python, the **for** loop is employed to iterate over a specified sequence, executing code repeatedly. In algorithms like this, the primary aim of the **for** loop is to implement particular code statements for each element within a sequence. The **for** keyword initiates the loop, followed by the loop variable (commonly denoted as "element") and the **in** keyword. This signifies iterating through the sequence **ip\_addresses** and assigning each value to the loop variable "element".

## Remove IP addresses that are on the remove list

To fulfill the requirements of my algorithm, it's necessary to eliminate any IP address from the **allow\_list (ip\_addresses)** that also appears in the **remove\_list.** Since **ip\_addresses** didn't contain any duplicates, I could accomplish this task using the following code:



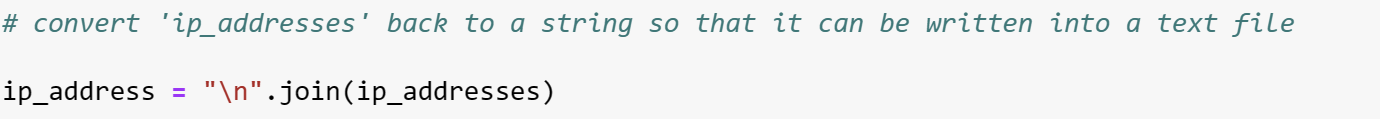
Initially, inside the for loop, I established a condition to determine whether the loop variable **"element"** was present in the **ip\_addresses** list. This precaution was taken to prevent errors from attempting to remove elements that weren't found in **ip\_addresses.** Subsequently, within that conditional block, I invoked the **.remove()** method on **ip\_addresses**, providing the loop variable **"element"** as the argument. This ensured that each IP address listed in the **remove\_list** would be successfully removed from **ip\_addresses.**

## 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

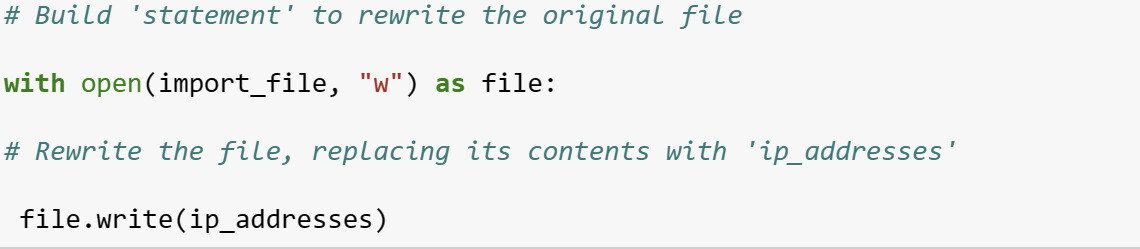
addresses. To do so, I first needed to convert the list back into a string. I used the .join()

method for this:

****

The **.join()** method merges all items within an iterable into a single string. It operates on a string containing characters that will separate the elements in the iterable when joined. In this particular algorithm, I employed the **.join()** method to form a string from the **ip\_addresses** list, which I later passed as an argument to the **.write()** method for writing to the file "allow\_list.txt". I utilized the string **"\n"** as the separator to prompt Python to place each element on a new line.

Subsequently, I employed another **with** statement along with the **.write()** method to update the file:

****

This time, I included a second argument of **"w"** when using the **open()** function within my **with** statement. This **"w"** argument signifies the intent to open a file for writing, thereby overwriting its contents. With this **"w"** argument, I'm able to invoke the **.write()** method within the **with** statement. The **.write()** method facilitates writing string data to a designated file, replacing any existing content.

In this scenario, my objective was to write the updated allow list as a string to the file **"allow\_list.txt"**. By doing so, access to restricted content would be revoked for any IP addresses removed from the allow list. To accomplish this, I appended the **.write()** method to the file object file identified in the with statement. I specified the **ip\_addresses** variable as the argument, indicating that the contents of the file specified in the with statement should be replaced with the data from this variable.

## Summary

I devised an algorithm aimed at removing IP addresses identified in a **remove\_list** variable from the "allow\_list.txt" file, which contains approved IP addresses. This algorithm encompassed several steps: first, opening the file and converting its contents into a string for reading. Subsequently, I converted this string into a list stored in the variable **ip\_addresses**.

Next, I iterated through the IP addresses in **remove\_list**, evaluating each element to determine if it belonged to the **ip\_addresses** list. Whenever a match was found, I utilized the **.remove()** method to eliminate the element from **ip\_addresses.** Following this, I employed the **.join()** method to convert **ip\_addresses** back into a string format, facilitating the overwriting of the contents of the "allow\_list.txt" file with the revised list of IP addresses.

This process ensures that any IP addresses listed in **remove\_list** are effectively removed from the allow list, maintaining the integrity of the access control mechanism.