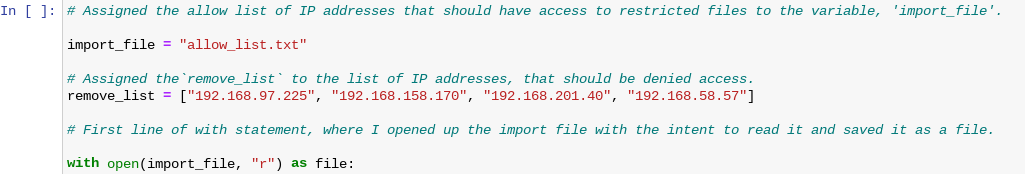
# Security Analyst**:** Christopher D’Angelantonio

# **Algorithm for file updates in Python**

## **Project description:**

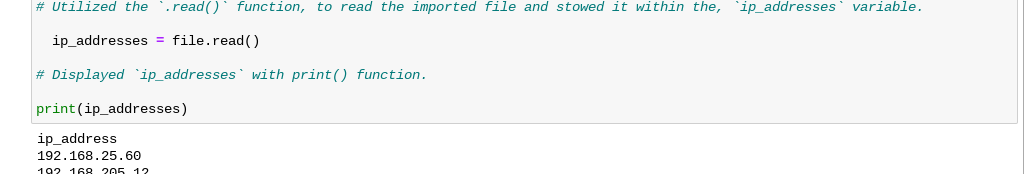
In this scenario, I occupy the role of a security analyst who has been tasked with designing multi-functional algorithms via leveraging the Python programming language in order to automate file parsing procedures, which conduct file access control maintenance and updates. Specifically, this algorithm will be used to parse IP addresses that belong to users who have approved clearance for accessing restricted files and to relinquish access from IP addresses that possess expired authorization.

## **Open the file that contains the allow list:**



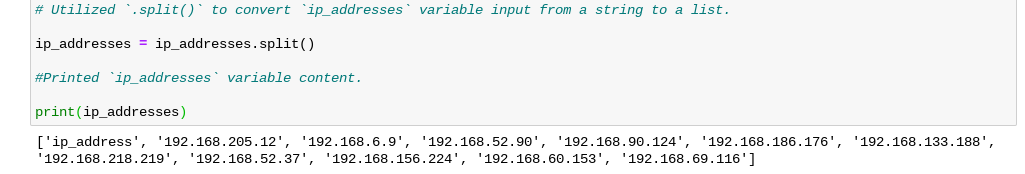
Firstly, I assigned the ‘allow\_list.txt’ file to the variable ‘import\_file’ using an equal operator. I then used the same equal assignment concept, to pair the string list of restricted IP addresses to a variable called, ‘remove\_list’. Next, I utilized the with statement to open the ‘import\_file’ and applied the “r” operator to the body of the statement to indicate that I would like to read the file. Notably, the as expression was used to save the ‘import\_file’s’ contents as a file while being in the with statement.

## **Read the file contents:**



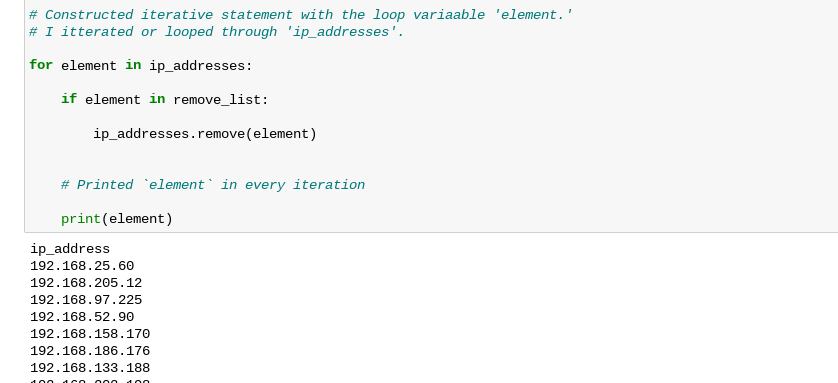
Furthermore, I then used the .read() function within the with statements inner conditional, to store the output of the import\_file’s read within the ip\_addresses variable in order to allow myself not to have to re-read the original ‘import\_file’ every time, I wish to call upon the data within the allow\_list.txt file. After that I printed the output of the ‘allow\_list.txt’ file via inputting its assigned variable ‘import file’ within the print() function parameters.

## **Convert the string into a list:**



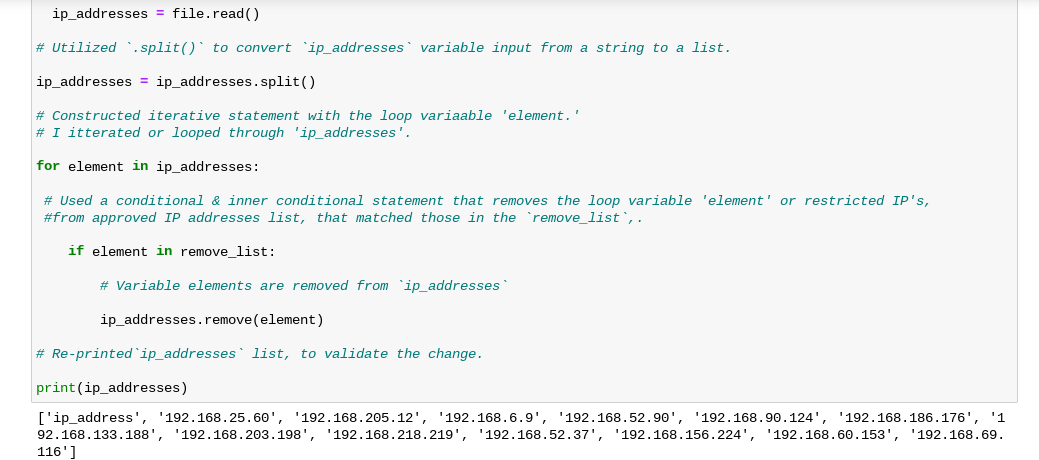
I then used the .split() to transmute the ‘ip\_addresses’ list from a string to a list notation and applied the print() function, to display the approved IP\_addresses’s files listed IP’s.

## **Iterate through the remove list:**



In this code block, I constructed an iterative statement via applying a looping a variable named ‘element’ against the approved import\_file IPs stored within the ip\_addresses list in order to run a conditional ‘if’ statement to iterate through the allowed list of IPs to see if any of IP’s within the IP address’s list match the variable element associated with the IPs in the ‘restricted\_list’.

**Remove IP addresses that are on the remove list:**



I then printed the remaining log data strings of authentically approved IP data, that were still residing within the ‘approve\_ ip\_addresses’ list in order to validate that the desired changes were made by the loop or the multifunctional algorithm. Additionally, I used the expression ‘ip\_addresses.remove(element)’ in order to run a remove() function within the loops inner conditional statement, that prompted the python interpreter to relinquish any falsely-approved IP’s within the ip\_addresses list, that match the elemental notations of those already deemed unauthorized to access restricted files within the restricted\_files list.

## **Update the file with the revised list of IP addresses:\\\\\**

In order to ensure that the correct execution of the prevention stage within the NIST CSF Lifecycle, I updated the ‘ip\_addresses’ list to convert it into a string, to be stowed in a text file format and I accomplished this through utilizing the ‘“”.join(ip\_addressses)’ expression. Following that step, I used a w statement to re-open the initial version of the import\_file along with a “w” operation and as file expression to indicate that I was opening the file with the intention of rewriting it as well as saving it while using the with statement. Afterward I used the file.write() to overwrite the unrevised ‘ip\_addresses list with the revised IP list contents and then used another with open() and read statement to re-open the newly drafted ip\_addresses list as a file followed by assigning this file’s read output to another variable named ‘text’. Lastly, I printed the updated file version of the ip\_addresses file by inputting the text variable assigned to it within the final print() function. Thus, at this point in the coding process I validated that the file was indeed programmed correctly and according to the current security standards for the organizations IP access control list as well as the conditional principles set within my code.

## **Summary:**

In this scenario, I used the Python programming language to leverage the multifunctional power of an algorithmic sparsing program that I built with the intent to make file access management along with the file access configurations that grant specific worker IPs access to organizational records to be more seal proof and reliable. Specifically, it's imperative that I scripted a multifunctional algorithm that was able to automate the task of determining what worker IPs rightfully belonged to the allow\_list.txt list and which of those needed to be removed as well as placed into the restricted\_list of IPs of workers who were meant to have that access revoked. Informatively, I perform these operations by utilizing a ‘for’ loop to iterate through an entire string sequence of IPs contained within the allow\_list and compare them against a loop variable called element used within a conditional ‘if’ statement, to compare whether any IPs match the numeral indices or character notations of the IPs contained in the restricted\_list and printed the results to visually compare IPs of both lists. I then used expression-based functions along with conditional operators, to remove these problematic IP’s and re-wrote the newly revised list of IPs over the old version of the import\_file to update the access\_list.tx file. Therefore, the Python coding language allows a security analyst to streamline a variety of security related tasks or operations via providing autonomous processing capabilities which both allows an analyst to concentrate their manual efforts on less tedious procedures, while allowing an organization to have a better peace of mind knowing that their file access control list is automatically and regularly updated according to the most current access parameters. Ultimately, automated operations like this as well as others found in cybersecurity automated playbooks, allow an organization to drastically increase the effectiveness of their security framework and strength of their security posture.