

# Algorithm for file updates in Python

## Project description

This project automates the process of updating the allow list file for restricted content at an organization. The allow list file, "allow\_list.txt", identifies the IP addresses that are allowed to access this content.

A separate remove list identifies IP addresses that should no longer have access. The algorithm in this project opens the allow list file, converts it to a list, and then iterates through the remove list. If an IP address is found in the remove list, it is removed from the allow list. The updated allow list is then written back to the file.

This algorithm can be used to automate the process of updating the allow list file, which can save time and reduce the risk of errors. It can also be used to ensure that only authorized IP addresses have access to restricted 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:

```
# Assign 'import_file' to the name of the file
import_file = "allow_list.txt"
```

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

```
# Build 'with' statement to read in the initial contents of the file
with open(import_file, "r") as file:
```

In my algorithm, I use the `with` statement to open the allow list file in read mode. This allows me to access the IP addresses stored in the file. The `with` statement will also close the file after I am finished with it, which helps to manage resources.

The code `with open(import_file, "r") as file:` has two parameters. The first parameter is the file name, and the second parameter is the mode. The mode tells Python how I want to open the file. In this case, I am opening the file in read mode, so I use the `'r'` mode.

The `as` keyword assigns the output of the `open()` function to the variable `file`. This means that I can use the `file` variable to access the contents of the file.

## Read the file contents

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

```
with open(import_file, "r") as file:

    # Use '.read()' to read the imported file and store it in a variable
    # named 'ip_addresses'
    ip_addresses = file.read()
```

When using an `open()` function that includes the argument `"r"` for “read,” I can call the `.read()` function in the body of the `with` statement. The `.read()` method converts the file into a string and allows me to read it. I applied the `.read()` method to the `file` variable identified in the `with` statement. Then, I assigned the string output of this method to the variable `ip_addresses`.

This code reads the contents of the `"allow_list.txt"` file into a string format that allows me to later use the string to organize and extract 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:

```
# Use '.split()' to convert 'ip_addresses' from a string to a list
ip_addresses = ip_addresses.split()
```

## 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:

```
# Build iterative statement
# Name loop variable 'element'
# Loop through 'remove_list'
for element in remove_list:
```

The `for` loop in Python repeats code for a specified sequence. The overall purpose of the `for` loop in a Python algorithm like this is to apply specific code statements to all elements in a sequence. The `for` keyword starts the `for` loop. It is followed by the loop variable `element` and the keyword `in`. The keyword `in` indicates to iterate through the sequence `ip_addresses` and assign each value to the loop variable `element`.

## Remove IP addresses that are on the remove list

```
for element in remove_list:

    # Create conditional statement to evaluate if 'element' is in
    'ip_addresses'

    if element in ip_addresses:

        # Use the '.remove()' method to remove elements from 'ip_addresses'

        ip_addresses.remove(element)
```

First, within my `for` loop, I created a conditional that evaluated whether or not the loop variable `element` was found in the `ip_addresses` list. I did this because applying `.remove()` to elements that were not found in `ip_addresses` would result in an error.

Then, within that conditional, I applied `.remove()` to `ip_addresses`. I passed in the loop variable `element` as the argument so that each IP address that was in the `remove_list` would be removed from `ip_addresses`.

```
# Define the 'remove_list' variable
remove_list = ["192.168.218.160", "192.168.97.225"]
```

Here, this line of code defines a variable called `remove_list`. The `remove_list` variable is a list of IP addresses that should be removed from the allow list. The list contains two IP addresses: `192.168.218.160` and `192.168.97.225`.

```
allow_list.txt
Python_Algorithm.py
4 192.168.96.200
5 192.168.247.153
6 192.168.3.252
7 192.168.116.187
8 192.168.15.110
9 192.168.39.246
10 192.168.218.160
11 192.168.97.225
```

## 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:

```
# Convert 'ip_addresses' back to a string so that it can be written into
the text file

ip_addresses = "\n".join(ip_addresses)
```

The `.join()` method combines all items in an iterable into a string. The `.join()` method is applied to a string containing characters that will separate the elements in the iterable once joined into a string.

In this algorithm, I used the `.join()` method to create a string from the list `ip_addresses` so that I could pass it in as an argument to the `.write()` method when writing to the file `"allow_list.txt"`. I used the string `("\\n")` as the separator to instruct Python to place each element on a new line.

Then, I used another `with` statement and the `.write()` method to update the file:

```
# Build 'with' statement to rewrite the original file

with open(import_file, "w") as file:

    # Rewrite the file, replacing its contents with 'ip_addresses'

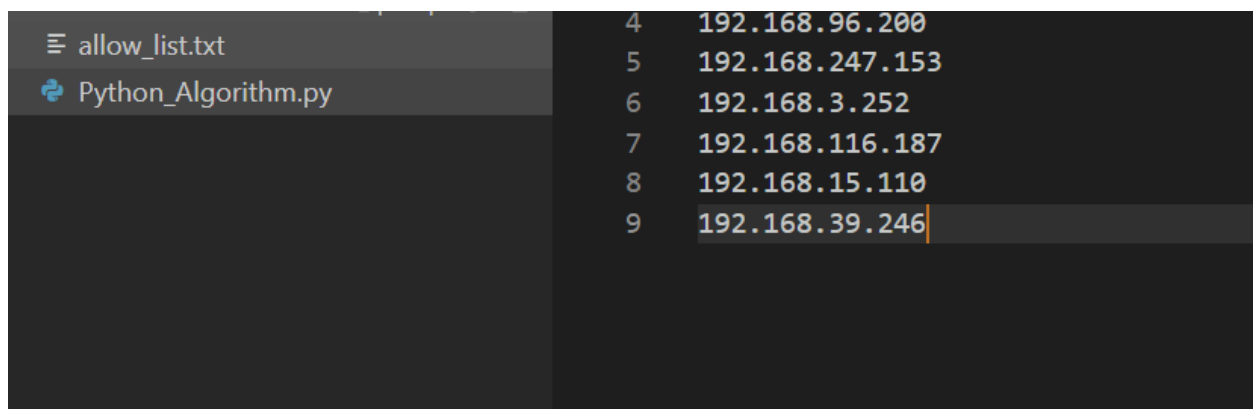
    file.write(ip_addresses)
```

I used a second argument of "w" with the `open()` function in my `with` statement. This argument indicates that I want to open a file to write over its contents. When using this argument "w", I can call the `.write()` function in the body of the `with` statement. The `.write()` function writes string data to a specified file and replaces any existing file content.

In this case I want to write the updated allow list as a string to the file `"allow_list.txt"`. This way, the restricted content will no longer be accessible to any IP addresses that were removed from the allow list.

To rewrite the file, I appended the `.write()` function to the file object `file` that I identified in the `with` statement. I passed in the `ip_addresses` variable as the argument to specify that the contents of the file specified in the `with` statement should be replaced with the data in this variable.

Now, let's run the script and see if our code does indeed remove those IP addresses:



The screenshot shows a code editor with a file named `allow_list.txt` open. The file contains a list of IP addresses, each on a new line, numbered 4 through 9. The IP addresses are: 192.168.96.200, 192.168.247.153, 192.168.3.252, 192.168.116.187, 192.168.15.110, and 192.168.39.246. The cursor is positioned at the end of the last line.

```
4 192.168.96.200
5 192.168.247.153
6 192.168.3.252
7 192.168.116.187
8 192.168.15.110
9 192.168.39.246
```

## Summary

This project automates the process of updating the allow list file for restricted content at an organization. The allow list file identifies the IP addresses that are allowed to access this content.

The algorithm in this project opens the allow list file, converts it to a list, and then iterates through the remove list. If an IP address is found in the remove list, it is removed from the allow list. The updated allow list is then written back to the file.

This algorithm can be used to save time and reduce the risk of errors in the allow list file. It can also be used to ensure that only authorized IP addresses have access to restricted content.

The benefits of this project include:

- Time savings: The algorithm can automate the process of updating the allow list file, which can save time and resources.
- Reduced errors: The algorithm can help to reduce the risk of errors in the allow list file.
- Improved security: The algorithm can help to improve the security of the organization's restricted content by ensuring that only authorized IP addresses have access.