Algorithm for file updates in Python by Chidubem Onochie-Onyetenu

Project description

I am required to regularly update a file that identifies employees who can access restricted content. Access to restricted content is controlled with an allow list of Internet Protocol (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 by removing those IP addresses.

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 the file name as a string to the import_file variable. Then, I used a with statement to open the file:

```
# Assign `import_file` to the name of the file

import_file = "allow_list.txt"

# Build `with` statement to read in the initial contents of the file

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

In my algorithm, the with statement is used with the open() function in read mode to open the allow list file so that I can read the IP addresses stored in it. The with keyword manages resources by closing the file after the with statement ends. In the code with open(import_file, "r") as file:, the open() function has two parameters: the first identifies the file to import, and the second, "r", indicates that I want to read it. The as keyword assigns the variable file, which stores the output of open() while I work within the with statement.

Read the file contents

To read the file contents, I used the .read() method to convert the file into a 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 open() is called with the argument "r", I can call the .read() method in the body of the with statement. The .read() method converts the file into a string. I applied it to the file variable from the with statement and assigned the string output to the variable ip_addresses.

In summary, this code reads the contents of the “allow_list.txt” file into a string, which lets me later organize and extract data in my Python program.

Convert the string into a list

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

Therefore, I 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()
```

The .split() method is called by appending it to a string variable, and it converts the string into a list. Splitting ip_addresses into a list makes it easier to remove individual IP addresses. By default, .split() separates text at whitespace, so in this algorithm it converts the string of whitespace-separated IP addresses into a list of IP addresses. I reassigned this list back to the variable ip_addresses.

Iterate through the allow list

A key part of my algorithm involves iterating through the IP addresses in ip_addresses so that each one can be checked against the remove list. To do this, I used a for loop:

```
# Build iterative statement
# Name loop variable `element`
# Loop through `ip_addresses`

for element in ip_addresses:
```

A for loop repeats code for each element in a sequence. The for keyword starts the loop, followed by the loop variable element and the keyword in, which indicates that the loop goes through the sequence ip_addresses and assigns each value to element.

Remove IP addresses that are on the remove list

My algorithm must remove any IP address in the allow list, ip_addresses, that is also in remove_list. Because ip_addresses contained no duplicates, I could use the following code:

```
for element in ip_addresses:

  # Build conditional statement
  # If current element is in `remove_list`,

    if element in remove_list:

        # then current element should be removed from `ip_addresses`

        ip_addresses.remove(element)
```

Within my for loop, I created a conditional that evaluates whether the loop variable element is in remove_list. Then, within that conditional, I applied .remove() to ip_addresses, passing in element as the argument, so that each IP address on the remove list is removed from ip_addresses.

Update the file with the revised list of IP addresses

As a final step, I needed to update the allow list file with the revised list of IP addresses. First, I converted the list back into a string using the .join() method:

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

ip_addresses = " ".join(ip_addresses)

# 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 = file.write(ip_addresses)
```

The .join() method combines all items in an iterable into a string. It is applied to a string containing the characters that will separate the elements once they are joined. In this algorithm, I used .join() to create a string from the list ip_addresses, with a space (" ") as the separator, so that I could pass the string to the .write() method when writing to “allow_list.txt”.

Then, I used another with statement and the .write() method to update the file. I used "w" as the second argument of open(), which opens the file so that its contents are written over. With "w", I can call .write() in the body of the with statement; it writes string data to the file and replaces any existing content. Here, I wrote the updated allow list to “allow_list.txt”, so the restricted content is no longer accessible to any IP addresses that were removed. I called .write() on the file object identified in the with statement and passed in ip_addresses as the argument.

Summary

I created an algorithm that removes the IP addresses in a remove_list variable from the “allow_list.txt” file of approved IP addresses. The algorithm opens the file, reads it into a string, and converts that string into a list stored in ip_addresses. It then iterates through ip_addresses, checking at each iteration whether the element is in remove_list. If it is, the .remove() method removes it from ip_addresses. Finally, the .join() method converts ip_addresses back into a string, which is written over the contents of “allow_list.txt” with the revised list.