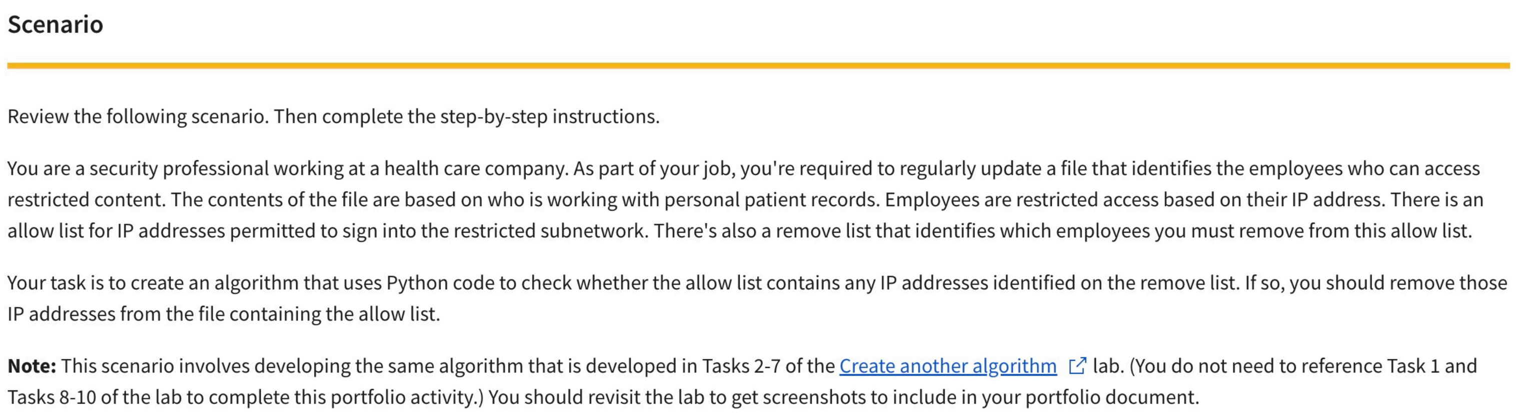
Updating Files - Python Algorithm

Introduction

This presentation covers three key aspects of the project: the scenario, the notes used, and the final result. The scenario outlines the project goals, the notes provide guidance, and the final project demonstrates how these elements were applied.

Scenario:



Scenario

Review the following scenario. Then complete the step-by-step instructions.

You are a security professional working at a health care company. As part of your job, you're required to regularly update a file that identifies the employees who can access restricted content. The contents of the file are based on who is working with personal patient records. Employees are restricted access based on their IP address. There is an allow list for IP addresses permitted to sign into the restricted subnetwork. There's also a remove list that identifies which employees you must remove from this allow list.

Your task is to create an algorithm that uses Python code to check whether the allow list contains any IP addresses identified on the remove list. If so, you should remove those IP addresses from the file containing the allow list.

Note: This scenario involves developing the same algorithm that is developed in Tasks 2-7 of the Create another algorithm lab. (You do not need to reference Task 1 and Tasks 8-10 of the lab to complete this portfolio activity.) You should revisit the lab to get screenshots to include in your portfolio document.

Final Project:

Update a file through a Python algorithm

Project description

At my organization, access to restricted content is controlled with an allow list of IP addresses. The "allow_list.txt" le identi es these IP addresses. A separate removal list identi es IP addresses that should no longer have access to this content. I created an algorithm to automate updating the "allow_list.txt" le and remove these IP addresses that should no longer have access.

Open the le that contains the allow list

For the rst part of the algorithm, I opened the `"allow_list.txt"` le. First, I assigned this le 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 le:

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

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

In my procedure, the with statement is utilized alongside the .open() function in read mode to access the allow list le for the purpose of reading it. The intention behind opening the le is to enable me to retrieve the IP addresses stored in the allow list le. The with keyword is employed to manage resources e ciently by automatically closing the le a er exiting the with statement. In the code with open(import_ le, "r") as le:, the open() function takes two arguments. The rst speci es the le to import, and the second indicates the desired operation with the le. In this instance, "r" signi es that I intend to read it. Furthermore, the code employs the as keyword to assign a variable named le; this variable stores the output of the .open() function while I operate within the with statement.

Read the le contents

In order to read the le 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 le into a string and allows me to read it. I applied the .read() method to the file variable identi ed in the with statement. Then, I assigned the string output of this method to the variable ip_addresses.

In summary, this code reads the contents of the "allow_list.txt" le 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:

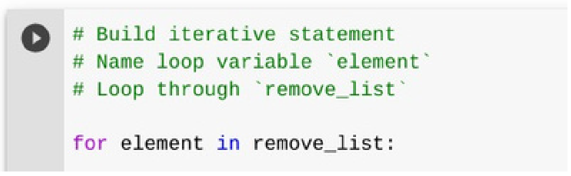
```
# Use `.split()` to convert `ip_addresses` from a string to a list

ip_addresses = ip_addresses.split()
```

The .split() function is called by appending it to a string variable. It works by converting the contents of a string to a list. The purpose of spli ing ip_addresses into a list is to make it easier to remove IP addresses from the allow list. By default, the .split() function splits the text by whitespace into list elements. In this algorithm, the .split() function takes the data stored in the variable ip_addresses, which is a string of IP addresses that are each separated by a whitespace, and it converts this string into a list of IP addresses. To store this list, I reassigned it back to the variable ip_addresses.

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 speci ed sequence. The overall purpose of the for loop in a Python algorithm like this is to apply speci c 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

My algorithm requires removing any IP address from the allow list, ip_addresses, that is also contained in remove_list. Because there were not any duplicates in ip_addresses, I was able to use the following code to do this:

```
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.

Update the le with the revised list of IP addresses

As a nal step in my algorithm, I needed to update the allow list le with the revised list of IP addresses. To do so, I rst 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 le "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 le:

```
# 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)
```

This time, I utilized a second argument of "w" with the open() function in my with statement. This argument signi es my intention to open a le for writing, thereby overwriting its existing contents. By employing this "w" argument, I was able to invoke the .write() function within the body of the with statement. The .write() function is responsible for writing string data to a designated le and replacing any pre existing content. In this instance, my objective was to compose the updated allow list as a string within the le "allow_list.txt". Consequently, the restricted content would become inaccessible to any IP addresses that had been removed from the allow list. To accomplish this, I appended the .write() function to the le object named ' le', which I identi ed within the with statement. I then supplied the ip_addresses variable as the argument, specifying that the contents of the le indicated in the with statement should be replaced with the data stored in this variable.

Summary

I devised an algorithm tasked with eliminating IP addresses identi ed in a remove_list variable from the "allow_list.txt" le containing approved IP addresses. This algorithm entailed opening the le, converting it to a string for reading, and then transforming this string into a list stored in the variable ip_addresses. Subsequently, I iterated through the IP addresses in remove_list. During each iteration, I assessed whether the element was present in the ip_addresses list. If found, I utilized the .remove() method to eliminate the element from ip_addresses. Following this, I utilized the .join() method to convert the ip_addresses back into a string, thereby facilitating the overwrite of the contents of the "allow_list.txt" le with the updated list of IP addresses.