Algorithm for file updates in Python

Project description

In this exercise, the task was to write an algorithm using the Python programming language which updates a file containing a list of IP addresses, having removed a subset of those addresses which should no longer be allowed to access the network or resource. Python is a general-purpose programming language, which means that it can be used to accomplish a wide range of different tasks.

Python can be used to automate long-running processes in cybersecurity, which would otherwise take a long time to perform manually by hand. This saves time as the programmer is now free to perform other tasks while the code runs.

Open the file that contains the allow list

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

# Assign `remove_list` to a list of IP addresses that are no longer allowed to access restricted information.
remove_list = ["192.168.97.225", "192.168.158.170", "192.168.201.40", "192.168.58.57"]

# First line of `with` statement
with open(import_file, "r") as file:
```

The `with` statement in Python is known as a file handler. It is used to open a file in a specified mode, provided as an argument to the `open()` function. The `open()` function in Python can take different values as the second argument, which specifies how the file should be opened. Values that the second argument can have include `r`, `w`, and `a`, as well as `r+`, `w+` and `a+`. Adding the `+` opens the file for both reading and writing. In this example, the file `allow_list.txt` is opened in read-only mode.

File handlers are structures that manage the opening and closing of files in Python. Using the `with open()` file handler ensures that the file will be closed automatically once the code finishes. If a file handler is not used, then the file has to be closed manually using the `close()` function, which, if forgotten, can cause unexpected behaviour.

Read the file contents

```
# Build `with` statement to read in the initial contents of the file
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()
# Display `ip_addresses`
print(ip_addresses)
```

Once the file has been opened using the file handler, it can be read to using the `read()` method. This method takes no arguments, but instead reads the contents of the file into a single string. In this example, the string contains the data inside the `allow_list.txt` file, as it has been assigned to the variable with the name `ip_addresses`. Printing the value of this variable using the `print()` function reveals the contents of the file, including any newline characters.

Convert the string into a list

```
  # Use `.read()` to read the imported file and store it in a variable named `ip_addresses`
  ip_addresses = file.read()
# Use `.split()` to convert `ip_addresses` from a string to a list
ip_addresses = ip_addresses.split()
# Display `ip_addresses`
print(ip_addresses)
```

Having read the file, it needs to be converted into a list so that it is possible to make modifications to existing data inside the file, instead of just adding new lines of text to the file. This is done using the `split()` method on the variable `ip_addresses`. In order to replace the original file with the new, parsed version of the information, the contents of the file is reassigned to the `ip_addresses` variable, which overwrites the original data with the new data.

The `split()` method takes an optional argument, which is the delimiter that is used to split the contents of the file. If no argument is supplied, the default delimiter of an empty space character is used to split the file. Using the `split()` method on a string converts it into a list of smaller strings, and the location(s) that the string is split at depends on the delimiter that, if any, is passed to the `split()` method.

Iterate through the remove list

```
ip_addresses = ip_addresses.split()

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

for element in ip_addresses:
    # Display `element` in every iteration
    print(element)
```

There are two ways to loop through an iterable in Python. The first way to do so is a `while` loop, while the second way, and the way that is used in this example, is a `for` loop. A `for` loop in Python takes a collection or iterable, and applies the body of the loop to each element in the collection. For loops take a counter and an iterable as parameters.

The first parameter is the loop variable. This is the name given to each element of the collection so that a transformation can be applied within the loop body. The second component is the iterable that you are looping over.

The `in` keyword tells Python to apply the transformation of the loop variable to each element of the collection supplied in the loop. In this example, we loop over the collection, and apply the `print()` function to each element, which displays it to the screen.

Remove IP addresses that are on the remove list

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

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)

# Display `ip_addresses`

print(ip_addresses)
```

The `remove()` method in Python is used to remove the first occurrence of an element from a list or other iterable. In the example, we use a for loop, described above, to loop through the list of IP addresses and check whether the current IP address is in the list of IP addresses that should be disallowed from accessing the network, using an `if` statement. If this is the case, we use the `remove()` method to delete that address from the list. Once the loop has finished executing, it will have removed all the IP addresses from the `ip_addresses` list that also occur

within the `remove_list`. The `remove()` method takes two parameters as input; the element to be removed from the iterable, and the iterable itself.

Update the file with the revised list of IP addresses

```
# 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.write(ip_addresses)
```

The `write()` method takes as an argument the string that you want to write into the file specified by the handler. The parameter to the `write()` function can also be a variable, but the type of the parameter passed to the `write()` function is required to be a string. If the data is of any other type, it must be converted into a string using the `str()` function. In order to be able to write to the file, the file must be opened using the `w` option, or one of the options that opens the file in both read and write mode, such as `w+` or `r+`. The example code uses the `w` option, which opens the file in write-only mode.

Summary

In this task, the goal was to take a list of IP addresses, and update it using Python, with the goal of denying access to a set of IP addresses that needed access removed. To accomplish this task, the addresses were read into a file, and parsed into a format that was easier to work with. The unfavourable addresses were then removed from the list of IPs, and the file was updated to reflect the changes made.