Lab - Using Python to List and Search Messages

Objectives

- **Ask for hard-coded or user entered access token.**
- **Add a URL for the API request.**
- **Add a message counter.**
- **Provide additional information from the JSON data.**
- **Summarize the search results.**
- **Execute the program.**

Background / Scenario

Python can be used to automate tasks that may not be possible in the application or the developer's API site. For example, a Python program could get a list of all messages in a room, find all messages that suit certain criteria, and format and print or delete those messages.

This lab introduces you to using existing Python code to search for specific text similar to what you did in a previous lab using Postman. You will complete portions of the code to create a working application that display all of the rooms associated with a user and allow the user to select a room and search for messages that contain a search string in the selected room.

Required Resources

- Webex Teams account
- Postman application
- Webex Teams desktop application
- Python 3 with IDLE
- Python code files

Step 1: Ask for hard-coded or user-entered access token.

a. Start **IDLE**.

b. Open the file **08_find-message-text.py**.

c. Go to **Student Step #1** in the code. Use the input method to ask the users if they want to use the hardcoded access token. Store the result in a variable.

d. Create an **if-else** structure to evaluate the user input variable. If the user answered with an "**n**" or "**N**" use another input method to prompt the user to enter an access token. Assign this value the **accessToken** variable.

e. Set the value of the **accessToken** variable to start with the string **"Bearer "** followed by the access token value.

f. In the **else:** part of the conditional, assign a string that concatenates the word "Bearer", followed by a space, with your access token value to the variable **accessToken**. This will be used as the hardcoded access token.

g. Refer to your work in the previous lab or to the solution file, if necessary.

Step 2: Add a URL for the API request.

a. Go to **Student Step #2** in the code and locate the following statement:

```
r = requests.get( "<!!!REPLACEME with URL for Webex Teams Rooms API!!!>",
                    headers = {"Authorization": accessToken}
                )
```

b. Replace **[Add URL for Webex Teams Rooms]** with the URL for the Webex Teams List Rooms API endpoint.

Step 3: Add a message counter.

a. Go to **Student Step #3a** in the code. It is close to the end of the program. To let the users know how many messages were found that contain the search text, add a message counter variable after the commented section of the text. Initialize the variable to 0. Call the variable **messageCounter**.

b. Go to Student **Step #3b** in the code. Add a statement that will increment the counter variable for each message that is found.

Step 4: Provide additional information from the JSON data.

a. Go to **Student Step #4** in the code. The code has been commented to assist you.

b. If the search text is found, the program will display “Found a message with <message text>”, followed by text from any messages that meet that criteria. Note that **"text"** is a key in the **message** variable. Other keys are also in that variable.

   To make this information more helpful, add code which also displays:

   - The email address of the person who created the message.
   - The date/time when the message was created.

   Add two lines of code to display messages that identify the values being displayed and then the values from the message variable that are required. Follow the example below in which the value of the **text** key of the message variable is displayed:

   ```
   print("Message: " + message["text"])
   ```

   To do this, you will need know which keys in the message variable contain the values that you need. To find the name of the required keys, list messages for any room using either Cisco Webex for Developers or Postman to view messages. The returned output will display the names of all the keys used for each message.

c. To display the keys for the email address and date/time (and the rest of the key/value pairs associated with the messages JSON data) remove the comments (#) from the print function shown below. This code can be found approximately 30 lines above step, "# Student Step #3a:".

```
################################################################################

# Data is converted to JSON format and held in jsonData

################################################################################

jsonData = resp.json()

################################################################################
# To see what is in the JSON data remove the # from the statements below:
#
print(
    json.dumps(
        jsonData,
        indent=4
    )
)
################################################################################
```

Step 5: Summarize the search results.

a. Go to **Student Step #5** in the code. As the **for** loop runs, the program displays each message that is found to contain the search string. After no more messages are found the **for** loop exits and returns to the main body of the program. Before the program terminates, we will display summary information about the results of the search.

b. There can be two outcomes for the search, either no messages will be found, or one or more messages may be found. You need to display an appropriate message for each condition. You can do this by creating an **if-else** statement in which you address with the first condition in the **if** portion and the second condition in **else** portion. For example, look at the pseudocode below:

if no messages are found:

display "no messages found" message

else:

display number of messages found message

You have the variable **messageCounter** which tells you how many messages have been found. Its initial value is 0, and it is incremented for every message that is found. If no messages have been found, then the value of the variable should be 0. This is the condition you can use for the **if** statement.

Create a conditional with the Boolean condition of the **messageCounter** variable equaling **0** as the **if** condition. If this condition is true, display a message that tells the user that no messages were found. For the **else** condition, print a message that tells the user how many messages were found using a text string and the value of the **messageCounter** variable.

Step 6: Execute the program.

a. Choose **Run** > **Run Module** to execute the program

b. Debug as necessary.

Common Python problems include:

- **Punctuation and symbols:** " ",' ', :, =,==, (), { }, [ ] are meaningful.
- **Paired symbols:** must match (open and close).
- **Indentation:** standard indentation is four (or multiples of four) spaces, control structures additional indentation.

c. The solution file is: **08_find-message-text_sol.py**