DevNet Associate (DEVASC v1.0) Skills Assessment Answers

itexamanswers.net/devnet-associate-devasc-v1-0-skills-assessment-answers.html

January 17, 2021

DEVASC v1.0 Skills Assessment

Objectives

- **Part 1: Collect Required API Documentation**
- **Part 2: Code a Webex Teams Bot**

Background / Scenario

NOTE: Instructor Note: The devasc-sa.py is required for this Skills Assessment. We strongly encourage you to make this script available to your students when they start the Skills Assessment. Copying and pasting the following script from a PDF or Word document can be problematic, introducing a variety of spacing and line break issues. In addition, there is a **devasc-sa_sol.py** version with comments for your use.

Python can be used to integrate API calls from different services. In this Skills Assessment, you will edit and modify Python code for a Webex Teams bot that integrates Webex, MapQuest, and ISS APIs. The code continuously checks for messages in a user specified Webex Teams room that begins with a forward slash (/) followed by a city name (e.g. /Washington, DC). When such a message is found, the city name is extracted and used with the MapQuest API service to get the GPS coordinates for the specified city. Next, thesecoordinates are used with the ISS API to find the date and time of the next time the ISS will fly over. Finally, a message is sent back to the Webex Teams room informing the user about the next flyover of the ISS for the queried city.

When completed, the Webex Teams bot will:

• Ask the user for their access token or to use the hard-coded access token.
• Display a list of the user's Webex Teams rooms.
• Ask the user which Webex Teams room to monitor for queries starting with "/".
• Extract the city name of a message starting with "/" (e.g. /Washington, DC -> Washington, DC).
• Request the latitude and longitude of a specified city using the MapQuest API.
• Request the next flyover date and time using the ISS API.
• Send the next ISS flyover information back to the specified Webex Teams room.

Required Resources

- 1 PC with operating system of your choice
- MapQuest and Webex Teams developer access and API keys
- Virtual Box or VMWare
- DEVASC Virtual Machine
- The **devasc-sa.py** student script.

Note: To protect application environments like Webex Teams from bots or malicious access attempts, most APIs rate limit availability. If you make a large number of the same API calls, your API call may be blocked for a specific amount of time. The timeout is usually less than 5 minutes.

Instructions

Part 1: Collect Required API Documentation

In this Part, you will collect information from the Webex, MapQuest, and ISS API documentation. This information will be required in Part 2 when you are coding the Webex Teams bot. You will also investigate the Python **time** library that you will use to convert epoch timestamps to a human readable date and time.

Note: We recommend that you do not provide links to students for two reasons: (1) links change, and (2) students need to be able to find API documentation. At the time this Skills Assessment was written, these were the links to Webex, MapQuest, and IIS API documentation:

- https://developer.webex.com/docs/api/getting-started (token and API documentation)
- https://developer.mapquest.com/user/me/profile (for My Keys)
- https://developer.mapquest.com/documentation/ (for API documentation)
- http://open-notify.org/Open-Notify-API/ISS-Location-Now/

Step 1: Launch the DEVASC VM.

Although you can complete this Skills Assessment in other environments, these instructions are for the DEVASC VM only. Instructions for other environments are not included.

Step 2: Investigate the documentation for Webex Teams rooms and message APIs.

a. Login to your developer account for Webex.

b. Locate and copy your personal access token. What is the lifetime for your token? **12 Hours**

c. Find the URL that will list all the rooms to which you belong. Record the HTTP method and URL:
GET https://webexapis.com/v1/rooms

d. Find the URL that list all the messages for a specified room. Record the HTTP method and URL:
GET https://webexapis.com/v1/messages

e. Find the URL that will create a message for a specified room. Record the HTTP method and URL.
POST https://webexapis.com/v1/messages

Step 3: Investigate the locations key for the MapQuest address API.

a. Login to your developer account for MapQuest.

b. Locate and copy your Consumer Key. When does your key expire? **No Expired**

The Webex Teams bot will use location data returned by a call to the MapQuest address API.

c. Open Chromium and paste in the following URL, replacing **your_api_key** with your MapQuest key:

https://www.mapquestapi.com/geocoding/v1/address?
key=your_api_key&location=Washington,DC

d. Notice that the MapQuest locations key includes keys for latitude and longitude for the location you entered. Record the **lat** and **lng** values returned by MapQuest for Washington, D.C in the code below.

```
{
   "info": {
       "statuscode": 0,
       "copyright": {
           "text": "© 2020 MapQuest, Inc.",
           "imageUrl": "http://api.mqcdn.com/res/mqlogo.gif",
           "imageAltText": "© 2020 MapQuest, Inc."
       },
       "messages": []
   },
   "options": {
       "maxResults": -1,
       "thumbMaps": true,
       "ignoreLatLngInput": false
   },
   "results": [
       {
            "providedLocation": {
                 "location": "Washington,DC"
       },
      "locations": [
          {
            "street": "",
            "adminArea6": "",
            "adminArea6Type": "Neighborhood",
            "adminArea5": "Washington",
            "adminArea5Type": "City",
            "adminArea4": "District of Columbia",
            "adminArea4Type": "County",
            "adminArea3": "DC",
            "adminArea3Type": "State",
            "adminArea1": "US",
            "adminArea1Type": "Country",
            "postalCode": "",
            "geocodeQualityCode": "A5XAX",
            "geocodeQuality": "CITY",
            "dragPoint": false,
            "sideOfStreet": "N",
            "linkId": "282772166",
            "unknownInput": "",
            "type": "s",
            "latLng": {
                "lat": 38.892062,
                "lng": -77.019912
            },
            "displayLatLng": {
                "lat": 38.892062,
                "lng": -77.019912
            },
<output omitted>
             }
          ]
```

```
        }
    ]
}
```

Step 4: Investigate the documentation for the ISS pass times API.

a. Search for “ISS API documentation” on the internet.

b. On the ISS API documentation website, click the API documentation for **ISS Pass Times**.

c. What are the two required parameters (called **query strings** on the website) for the ISS pass times API?
lat for latitude and **lon** for longitude

d. What are the optional parameters for the ISS pass times API?
alt for altitude and **n** for number of passes to return

e. What is the URL for the ISS pass times API?
http://api.open-notify.org/iss-pass.json?lat=LAT&lon=LON

Step 5: Investigate the response key for the ISS pass times API.

a. Open **Postman** and create a new **Untitled Request**.

b. Paste in the ISS pass times URL.

c. Replace the latitude and longitude values with the values for Washington, D.C.

d. Click **Send**. You should get output similar to the following, although your time values will be different. By default, the ISS pass times API returns the next 5 passes over the specified location.

```
{
  "message": "success",
  "request": {
    "altitude": 100,
    "datetime": 1592669962,
    "latitude": 38.892062,
    "longitude": -77.019912,
    "passes": 5
  },
  "response": [
    {
      "duration": 602,
      "risetime": 1592672814
    },
    {
      "duration": 649,
      "risetime": 1592678613
    },
    {
      "duration": 423,
      "risetime": 1592684494
    },
    {
      "duration": 475,
      "risetime": 1592733041
    },
    {
      "duration": 651,
      "risetime": 1592738727
    }
  ]
}
```

Step 6: Investigate epoch timestamps and how to convert them to a human readable format.

In Part 2, you will use the **ctime** function of the Python **time** library to convert epoch time into a human readable date and time. That date and time will then be incorporated in a message that the Webex Teams bot posts to a room.

Search the internet for documentation of the Python **time** library to answer the following questions.
Preferably, you should review documentation from python.org although answers can be found elsewhere.

a. In relation to computer time, what does the term “epoch” mean?
The epoch is the point where the time starts and is platform dependent.

b. What function of the **time** library will return the epoch time on a given platform?
time.gmtime(0)

c. You can see the year, month, day, hour, and so on for the start of the epoch with the following Python code. Open a terminal, start Python 3, import the time library, and then replace <function> with the function you found above.

```
devasc@labvm:~$ python3
Python 3.8.2 (default, Apr 27 2020, 15:53:34)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import time
>>> print(str(time.gmtime(0)))
time.struct_time(tm_year=1970, tm_mon=1, tm_mday=1, tm_hour=0, tm_min=0, tm_sec=0,
tm_wday=3, tm_yday=1, tm_isdst=0)
>>>
```

d. The DEVASC VM epoch start time is the same as for Unix. When does the epoch start?
January 1st, 1970 at 00:00:00 UTC

e. The **risetime** in Step 4 is expressed in the number of seconds since the epoch time. What time function will convert the **risetime** to a human readable format?
time.ctime([secs])

f. What is the date for the first risetime in Step 4?

```
>>> print(str(time.strfime('%a, %d %b %Y %H:%M:%S', time.localtime(1592672814))))
<Date and Time is printed here>
>>>

>>> print(str(time.ctime(1592672814)))
Sat Jun 20 17:06:54 2020
>>>
```

Part 2: Code a Webex Teams Bot

Note: You will need the **devasc-sa.py** script open and ready to edit for this Part. Obtain this script from your instructor.

In this Part, you will use your understanding of Python and REST APIs along with the API documentation you gathered in Part 1 to finish the code for the Webex Teams bot.

Hint: Save the original file as a different name in case you need to start over. Comment out the code you are not yet testing. As you work through the following steps, save and run your script often. Use temporary print statements to verify that your variables contain the data you expect.

Step 1: Import libraries for API requests, JSON formatting, and epoch time conversion.

```
import requests
import json
import time
```

Step 2: Complete the if statement to ask the user for the Webex access token.

For this step, you are provided with the first user prompt and the **else** portion of an **if/else** statement. You need to code the **if** portion of the statement that will execute if the user says "N" or "n" to using the hardcoded access token. The **if** statement checks the value of **choice**, then if "N" or "n" are entered, asks the user for the value of the token. The user-entered value is then stored in the **accessToken** variable. The **accessToken** variable should be constructed just like the version of the **else** statement.

```
if choice == "N" or choice == "n":
    accessToken = input("What is your access token? ")
    accessToken = "Bearer " + accessToken
```

Step 3: Provide the URL to the Webex Teams room API.

Use your documentation from Part 1 to specify the correct Webex Teams room API that will return a list of the rooms that you are a member of and store them in the **r** variable.

```
r = requests.get(   "https://webexapis.com/v1/rooms",
                    headers = {"Authorization": accessToken}
                )
```

Step 4: Finish the loop to print the type and title of each room.

The list of rooms stored in the **r** variable is converted to JSON and stored in the **rooms** variable. Add a print statement that will display each room type and title.
The student can choose to format the statement however they wish as long as both type and title are printed.

```
for room in rooms:
    print ("Type: '" + room["type"] + "' Name: " + room["title"])
```

Step 5: Provide the URL to the Webex Teams messages API.

Use your documentation from Part 1 to specify the correct Webex Teams messages API. Every second, the bot will make a call to this API and print the latest message posted to the room.

```
r = requests.get("https://webexapis.com/v1/messages",
                         params = GetParameters,
                         headers = {"Authorization": accessToken}
```

Step 6: Provide your MapQuest API consumer key.

You documented this key in Part 1.
The student must use the consumer key, not the secret key.

Step 7: Provide the URL to the MapQuest address API.

The student was given this URL in Part 1.

```
r = requests.get("https://www.mapquestapi.com/geocoding/v1/address?",
                             params = mapsAPIGetParameters
                        )
```

Step 8: Provide the MapQuest key values for latitude and longitude.

Use your documentation in Part 1 to specify the correct format to store the values for the latitude and longitude keys.

```
locationLat = json_data["results"][0]["locations"][0]["displayLatLng"]["lat"]
locationLng = json_data["results"][0]["locations"][0]["displayLatLng"]["lng"]
```

Step 9: Provide the URL to the ISS pass times API.

Use your documentation from Part 1 to specify the correct ISS pass times API.

```
r = requests.get("http://api.open-notify.org/iss-pass.json",
                      params = issAPIGetParameters
                )
```

Step 10: Provide the ISS key values for risetime and duration.

Use your documentation in Part 1 to specify the correct format to store the values for the risetime and duration keys.

```
risetimeInEpochSeconds = json_data["response"][0]["risetime"]
durationInSeconds      = json_data["response"][0]["duration"]
```

Step 11: Convert the risetime epoch value to a human readable date and time.

In Part 1, you researched the **time** library. Use your knowledge to convert the epoch value for **risetime** into a human readable data and time format.

```
risetimeInFormattedString = str(time.ctime(risetimeInEpochSeconds))
```

Step 12: Complete the code to format the response message.

Use the variables that have been specified to format the response message that will be sent to the Webex Teams room. For example, a posted message in the room would look like the following where the location, risetime, and duration are shown in bold.

In **Austin**, **Texas** the ISS will fly over on **Thu Jun 18 18:42:36 2020** for **242** seconds.

```
responseMessage = "In {} the ISS will fly over on {} for {}
seconds.".format(locationResults, risetimeInFormattedString, durationInSeconds)
```

Step 13: Complete the code to post the message to the Webex Teams room.

The final step in the Webex Teams bot program is to format the API POST message that will send the **responseMessage** to the Webex Teams room. Provide each of the required variables and the URL for the Webex Teams message API to complete the code.

```
HTTPHeaders = {
              "Authorization": accessToken,
              "Content-Type": "application/json"
              }
PostData = {
              "roomId": roomIdToGetMessages,
              "text": responseMessage
            }
r = requests.post( "https://webexapis.com/v1/messages",
                              data = json.dumps(PostData),
                              headers = HTTPHeaders
                  )
```

Step 14: Run your program, test it, and troubleshoot as necessary.

a. In your Webex Teams client, create a room with a name of your choice, such as **My DEVASC SA Room**.

b. Post a message, such as **Hello room!**, to populate the room with at least one message.

c. Run your program and choose **My DEVASC SA Room** as the room the Webex Teams bot will monitor.

d. In the My DEVASC SA Room, post a location in the format **/location**. Messages that begin with a forward slash initiate the Webex Team bot to do its work. For example, the following should occur when Austin, Texas is entered.

In a terminal window:

```
devasc@labvm:~/$ python3 devasc-sa_sol.py
Do you wish to use the hard-coded Webex token? (y/n) y
List of rooms:
Type: 'group' Name: My DEVASC SA Room
<rest of rooms listed>
Which room should be monitored for /location messages? My DEVASC SA Room
Found rooms with the word My DEVASC SA Room
My DEVASC SA Room
Found room : My DEVASC SA Room
Received message: Hello room!
Received message: Hello room!
Received message: Hello room!
Received message: Hello room!
<continues to print every 1 second>
```

In Webex Teams My DEVASC SA Room, add a location.

/Austin, Texas

In the terminal window, the following prints:

```
Received message: /Austin, Texas
Location: Austin, Texas
Location GPS coordinates: 30.264979, -97.746598
Sending to Webex Teams: In Austin, Texas the ISS will fly over on Sat Jun 20 20:18:29
2020 for 645 seconds.
Received message: In Austin, Texas the ISS will fly over on Sat Jun 20 20:18:29 2020
for 645 seconds.
Received message: In Austin, Texas the ISS will fly over on Sat Jun 20 20:18:29 2020
for 645 seconds.
Received message: In Austin, Texas the ISS will fly over on Sat Jun 20 20:18:29 2020
for 645 seconds.
Received message: In Austin, Texas the ISS will fly over on Sat Jun 20 20:18:29 2020
for 645 seconds.
<continues to print every 1 second>
```

In Webex Teams, to following displays.

In Austin, Texas the ISS will fly over on Sat Jun 20 20:18:29 2020 for 645 seconds.

e. In your terminal window, enter **Ctrl+C** to exit the program.

Student Script

Below is the content of the **devasc-sa.py** script. However, we recommend that you use the **devasc-sa.py** file you obtain from your instructor. Copying and pasting the following script from a PDF or Word document can be problematic.

WARNING: You are responsible for correcting the code formatting errors if you choose to copy and paste the following

```
###############################################################
# This program:
# - Asks the user to enter an access token or use the hard coded access token.
# - Lists the user's Webex Teams rooms.
# - Asks the user which Webex Teams room to monitor for "/location" requests.
# - Monitors the selected Webex Team room every second for "/location" messages.
# - Discovers GPS coordinates for the "location" using MapQuest API.
# - Discovers the date and time of the next ISS flyover over the "location" using the
ISS API
# - Formats and sends the results back to the Webex Team room.
#
# The student will:
# 1. Import libraries for requests, json, and time.
# 2. Complete the if statement to ask the user for the Webex access token.
# 3. Provide the URL to the Webex Teams room API.
# 4. Create a loop to print the type and title of each room.
# 5. Provide the URL to the Webex Teams messages API.
# 6. Provide your MapQuest API consumer key.
# 7. Provide the URL to the MapQuest address API.
# 8. Provide the MapQuest key values for latitude and longitude.
# 9. Provide the URL to the ISS pass times API.
# 10. Provide the ISS key values risetime and duration.
# 11. Convert the risetime epoch value to a human readable date and time.
# 12. Complete the code to format the response message.
# 13. Complete the code to post the message to the Webex Teams room.
###############################################################

# 1. Import libraries for requests, json, and time.

<!!!REPLACEME with code for libraries>

# 2. Complete the if statement to ask the user for the Webex access token.
choice = input("Do you wish to use the hard-coded Webex token? (y/n) ")

<!!!REPLACEME with if statements to ask user for the Webex Teams Access Token!!!>
else:
    accessToken = "Bearer <!!!REPLACEME with hard-coded token!!!>"

# 3. Provide the URL to the Webex Teams room API.
r = requests.get(   "<!!!REPLACEME with URL!!!>",
                    headers = {"Authorization": accessToken}
                )

#####################################################################################
# DO NOT EDIT ANY BLOCKS WITH r.status_code
if not r.status_code == 200:
    raise Exception("Incorrect reply from Webex Teams API. Status code: {}. Text:
{}".format(r.status_code, r.text))
######################################################################################


# 4. Create a loop to print the type and title of each room.
```

```
print("List of rooms:")
rooms = r.json()["items"]
for room in rooms:
    <!!!REPLACEME with print code to finish the loop>

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

# SEARCH FOR WEBEX TEAMS ROOM TO MONITOR
#  - Searches for user-supplied room name.
#  - If found, print "found" message, else prints error.
#  - Stores values for later use by bot.
# DO NOT EDIT CODE IN THIS BLOCK
########################################################################################


while True:
    roomNameToSearch = input("Which room should be monitored for /location messages?
")
    roomIdToGetMessages = None

    for room in rooms:
        if(room["title"].find(roomNameToSearch) != -1):
            print ("Found rooms with the word " + roomNameToSearch)
            print(room["title"])
            roomIdToGetMessages = room["id"]
            roomTitleToGetMessages = room["title"]
            print("Found room : " + roomTitleToGetMessages)
            break

    if(roomIdToGetMessages == None):
        print("Sorry, I didn't find any room with " + roomNameToSearch + " in it.")
        print("Please try again...")
    else:
        break

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

# WEBEX TEAMS BOT CODE
#  Starts Webex bot to listen for and respond to /location messages.
########################################################################################


while True:
    time.sleep(1)
    GetParameters = {
                            "roomId": roomIdToGetMessages,
                            "max": 1
                    }
# 5. Provide the URL to the Webex Teams messages API.
    r = requests.get("<!!!REPLACEME with URL!!!>",
                         params = GetParameters,
                         headers = {"Authorization": accessToken}
```

```
                        )

    if not r.status_code == 200:
        raise Exception( "Incorrect reply from Webex Teams API. Status code: {}.
Text: {}".format(r.status_code, r.text))

    json_data = r.json()
    if len(json_data["items"]) == 0:
        raise Exception("There are no messages in the room.")

    messages = json_data["items"]
    message = messages[0]["text"]
    print("Received message: " + message)

    if message.find("/") == 0:
        location = message[1:]
# 6. Provide your MapQuest API consumer key.
        mapsAPIGetParameters = {
                                "location": location,
                                "key": "<!!!REPLACEME with your MapQuest API Key!!!>"
                               }
# 7. Provide the URL to the MapQuest address API.
        r = requests.get("<!!!REPLACEME with URL!!!>",
                             params = mapsAPIGetParameters
                        )
        json_data = r.json()

        if not json_data["info"]["statuscode"] == 0:
            raise Exception("Incorrect reply from MapQuest API. Status code:
{}".format(r.statuscode))

        locationResults = json_data["results"][0]["providedLocation"]["location"]
        print("Location: " + locationResults)

# 8. Provide the MapQuest key values for latitude and longitude.
        locationLat = json_data["<!!!REPLACEME!!!> with path to latitude key!!!>"]
        locationLng = json_data["<!!!REPLACEME!!!> with path to longitude key!!!>"]
        print("Location GPS coordinates: " + str(locationLat) + ", " +
str(locationLng))

        issAPIGetParameters = {
                                "lat": locationLat,
                                "lon": locationLng
                              }
# 9. Provide the URL to the ISS pass times API.
        r = requests.get("<!!!REPLACEME with URL!!!>",
                             params = issAPIGetParameters
                        )

        json_data = r.json()

        if not "response" in json_data:
            raise Exception("Incorrect reply from open-notify.org API. Status code: 
{}. Text: {}".format(r.status_code, r.text))

# 10. Provide the ISS key values risetime and duration.
        risetimeInEpochSeconds = json_data["<!!!REPLACEME!!!> with path to risetime 
key!!!>"]
        durationInSeconds      = json_data["<!!!REPLACEME!!!> with path to duration 
key!!!>"]

# 11. Convert the risetime epoch value to a human readable date and time.
        risetimeInFormattedString = <!!!REPLACEME with conversion code!!!>

# 12. Complete the code to format the response message.
#     Example responseMessage result: In Austin, Texas the ISS will fly over on Thu 
Jun 18 18:42:36 2020 for 242 seconds.
        responseMessage = "In {} the ISS will fly over on {} for {} 
seconds.".format(<!!!REPLACEME with required variables!!!>)

        print("Sending to Webex Teams: " +responseMessage)

# 13. Complete the code to post the message to the Webex Teams room.
        HTTPHeaders = {
                             "Authorization": <!!!REPLACEME!!!>,
                             "Content-Type": "application/json"
                           }
        PostData = {
                            "roomId": <!!!REPLACEME!!!>,
                            "text": <!!!REPLACEME!!!>
                        }

        r = requests.post( "<!!!REPLACEME with URL!!!>",
                              data = json.dumps(<!!!REPLACEME!!!>),
                              headers = <!!!REPLACEME!!!>
                         )
        if not r.status_code == 200:
            raise Exception("Incorrect reply from Webex Teams API. Status code: {}. 
Text: {}".format(r.status_code, r.text))
```

Answer Script

```
###############################################################
# INSTRUCTOR NOTE:
#  To demonstrate this program, hard-code your
#  Webex and MapQuest keys in steps 2 and 6.
###############################################################
# This program:
# - Asks the user to enter an access token or use the hard coded access token.
# - Lists the user's Webex Teams rooms.
# - Asks the user which Webex Teams room to monitor for "/location" requests.
# - Monitors the selected Webex Team room every second for "/location" messages.
# - Discovers GPS coordinates for the "location" using MapQuest API.
# - Discovers the date and time of the next ISS flyover over the "location" using the
ISS API
# - Formats and sends the results back to the Webex Team room.
#
# The student will:
# 1. Import libraries for requests, json, and time.
# 2. Complete the if statement to ask the user for the Webex access token.
# 3. Provide the URL to the Webex Teams room API.
# 4. Create a loop to print the type and title of each room.
# 5. Provide the URL to the Webex Teams messages API.
# 6. Provide your MapQuest API consumer key.
# 7. Provide the URL to the MapQuest address API.
# 8. Provide the MapQuest key values for latitude and longitude.
# 9. Provide the URL to the ISS pass times API.
# 10. Provide the ISS key values risetime and duration.
# 11. Convert the risetime epoch value to a human readable date and time.
# 12. Complete the code to format the response message.
# 13. Complete the code to post the message to the Webex Teams room.
###############################################################

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

# 1. Import libraries for requests, json, and time.

import requests
import json
import time

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

# 2. Complete the if statement to ask the user for the Webex access token.
#     Ask the user to use either the hard-coded token (access token within the code)
#     or for the user to input their access token.
#     Assign the hard-coded or user-entered access token to the variable accessToken.

choice = input("Do you wish to use the hard-coded Webex token? (y/n) ")

if choice == "N" or choice == "n":
  accessToken = input("What is your access token? ")
  accessToken = "Bearer " + accessToken
else:
    accessToken = "Bearer <!!!REPLACEME with hard-coded token!!!>"

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

# 3. Provide the URL to the Webex Teams room API.
#     Using the requests library, create a new HTTP GET Request to the Webex Teams
API
#     endpoint for Webex Teams Rooms. The local object "r" will hold the returned
data.

r = requests.get(   "https://webexapis.com/v1/rooms",
                    headers = {"Authorization": accessToken}
                )
######################################################################################

# DO NOT EDIT ANY BLOCKS WITH r.status_code
if not r.status_code == 200:
    raise Exception("Incorrect reply from Webex Teams API. Status code: {}. Text:
{}".format(r.status_code, r.text))

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

# 4. Create a loop to print the type and title of each room.
#     Displays a list of rooms.

print("List of rooms:")
rooms = r.json()["items"]
for room in rooms:
    print ("Type: " + room["type"] + ", Name: " + room["title"])

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

# SEARCH FOR WEBEX TEAMS ROOM TO MONITOR
#  - Searches for user-supplied room name.
#  - If found, print "found" message, else prints error.
#  - Stores values for later use by bot.
# DO NOT EDIT CODE IN THIS BLOCK
######################################################################################


while True:
    # Input the name of the room to be searched
    roomNameToSearch = input("Which room should be monitored for /location messages?
")

    # Defines a variable that will hold the roomId
    roomIdToGetMessages = None

    for room in rooms:
        # Searches for the room "title" using the variable roomNameToSearch
        if(room["title"].find(roomNameToSearch) != -1):
            # Displays the rooms found using the variable roomNameToSearch
(additional options included)
            print ("Found rooms with the word " + roomNameToSearch)
            print(room["title"])

            # Stores room id and room title into variables
            roomIdToGetMessages = room["id"]
            roomTitleToGetMessages = room["title"]
            print("Found room : " + roomTitleToGetMessages)
            break

    if(roomIdToGetMessages == None):
        print("Sorry, I didn't find any room with " + roomNameToSearch + " in it.")
        print("Please try again...")
    else:
        break

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

# WEBEX TEAMS BOT CODE
#  Starts Webex bot to listen for and respond to /location messages.
######################################################################################


while True:
    # always add 1 second of delay to the loop to not go over a rate limit of API
calls
    time.sleep(1)

    # the Webex Teams GET parameters
    #  "roomId" is the ID of the selected room
    #  "max": 1  limits to get only the very last message in the room
    GetParameters = {
                            "roomId": roomIdToGetMessages,
                            "max": 1
                         }
# 5. Provide the URL to the Webex Teams messages API.
    # Send a GET request to the Webex Teams messages API.
  # - Use the GetParameters to get only the latest message.
  # - Store the message in the "r" variable.
    r = requests.get("https://webexapis.com/v1/messages",
                         params = GetParameters,
                         headers = {"Authorization": accessToken}
                    )
    # verify if the retuned HTTP status code is 200/OK
    if not r.status_code == 200:
        raise Exception( "Incorrect reply from Webex Teams API. Status code: {}.
Text: {}".format(r.status_code, r.text))

    # get the JSON formatted returned data
    json_data = r.json()
    # check if there are any messages in the "items" array
    if len(json_data["items"]) == 0:
        raise Exception("There are no messages in the room.")

    # store the array of messages
    messages = json_data["items"]
    # store the text of the first message in the array
    message = messages[0]["text"]
    print("Received message: " + message)

    # check if the text of the message starts with the magic character "/" followed
by a location name
    #  e.g.  "/San Jose"
    if message.find("/") == 0:
        # name of a location (city) where we check for GPS coordinates using the
MapQuest API
        #  message[1:]  returns all letters of the message variable except the first
"/" character
        #   "/San Jose" is turned to "San Jose" and stored in the location variable
        location = message[1:]

# 6. Provide your MapQuest API consumer key.
        # MapQuest API GET parameters:
        # - "location" is the the location to lookup
        # - "key" is the Consumer Key you generated at
https://developer.mapquest.com/user/me/apps
        mapsAPIGetParameters = {
                                "location": location,
                                "key": "<!!!REPLACEME with your MapQuest API Key!!!>"
# MapQuest API key here
                               }
# 7. Provide the URL to the MapQuest address API.
        # Get location information using the MapQuest API geocode service using the
HTTP GET method
        r = requests.get("https://www.mapquestapi.com/geocoding/v1/address?",
                             params = mapsAPIGetParameters
                        )
        # Verify if the returned JSON data from the MapQuest API service are OK
        json_data = r.json()
        # check if the status key in the returned JSON data is "0"
        if not json_data["info"]["statuscode"] == 0:
            raise Exception("Incorrect reply from MapQuest API. Status code:
{}".format(r.statuscode))

        # store the location received from the MapQuest API in a variable
        locationResults = json_data["results"][0]["providedLocation"]["location"]
        # print the location address
        print("Location: " + locationResults)

# 8. Provide the MapQuest key values for latitude and longitude.
        # Set the lat and lng key as retuned by the MapQuest API in variables
        locationLat = json_data["results"][0]["locations"][0]["displayLatLng"]["lat"]
        locationLng = json_data["results"][0]["locations"][0]["displayLatLng"]["lng"]
        # print the location address
        print("Location GPS coordinates: " + str(locationLat) + ", " +
str(locationLng))

        # ISS flyover Documentation: http://open-notify-
api.readthedocs.io/en/latest/iss_pass.html
        # the ISS flyover API GET parameters
        #  "lat" is the latitude of the location
        #  "lon" is the longitude of the location
        issAPIGetParameters = {
                                "lat": locationLat,
                                "lon": locationLng
                              }
# 9. Provide the URL to the ISS pass times API.
        # Get IIS flyover information for the specified GPS coordinates.
        r = requests.get("http://api.open-notify.org/iss-pass.json",
                             params = issAPIGetParameters
                        )
        # Format the returned data as JSON.
        json_data = r.json()
        # Verify the returned JSON data containes the response key. If not, display
error.
        if not "response" in json_data:
            raise Exception("Incorrect reply from open-notify.org API. Status code:
{}. Text: {}".format(r.status_code, r.text))

# 10. Provide the ISS key values risetime and duration.
        # Store the risetime and duration of the first flyover in variables.
        risetimeInEpochSeconds = json_data["response"][0]["risetime"]
        durationInSeconds      = json_data["response"][0]["duration"]

# 11. Convert the risetime epoch value to a human readable date and time.
        # Use the time.ctime function to convert the risetime to a human readable
date and time.
        risetimeInFormattedString = str(time.ctime(risetimeInEpochSeconds))

# 12. Complete the code to format the response message.
#     Example responseMessage result: In Austin, Texas the ISS will fly over on Thu
Jun 18 18:42:36 2020 for 242 seconds.
        responseMessage = "In {} the ISS will fly over on {} for {}
seconds.".format(locationResults, risetimeInFormattedString, durationInSeconds)
        # print the response message
        print("Sending to Webex Teams: " +responseMessage)

# 13. Complete the code to post the message to the Webex Teams room.
        # the Webex Teams HTTP headers, including the Authoriztion and Content-Type
        HTTPHeaders = {
                             "Authorization": accessToken,
                             "Content-Type": "application/json"
                           }
        # The Webex Teams POST JSON data
        # - "roomId" is is ID of the selected room
        # - "text": is the responseMessage assembled above
        PostData = {
                            "roomId": roomIdToGetMessages,
                            "text": responseMessage
                        }
        # Post the call to the Webex Teams message API.
        r = requests.post( "https://webexapis.com/v1/messages",
                              data = json.dumps(PostData),
                              headers = HTTPHeaders
                         )
        if not r.status_code == 200:
            raise Exception("Incorrect reply from Webex Teams API. Status code: {}.
Text: {}".format(r.status_code, r.text))
```

Download devasc-sa.py and devasc-sa_sol.py files:

DEVASC v1.0 Skills Assessment Answers Python Files

1 file(s)   5.51 KB

Download