Lab – Create a Host Inventory in Python

Objectives

Part 1: Using Postman to Get a Network Host Inventory

Part 2: Using Python to Get a Network Host Inventory

Background / Scenario

In a production network, the hosts on the network are constantly changing. From a security point of view, it is very useful to know about these hosts and where they connect. The APIC-EM keeps track of the hosts that are connected on the network. This information can be periodically collected and processed by Python. In this lab, you will create a function that displays an inventory of hosts that can be processed by other Python programs.

First, you will learn about the requirements for making the host inventory request by using Postman. You will review the structure of the JSON that comprises the inventory. Then, you will replicate the process in a Python program.

Required Resources

- Postman
- Python 3 with IDLE
- Python **requests** module
- Python **tabulate** module
- The functions file that you have created or the **apic_em_functions_sol.py** file
- Access to the Internet

Note: Use the APIC-EM sandbox URL and credentials provided by your instructor. Only use the public APIC-EM URL and credentials for additional study after the conclusion of the workshop. For example purposes, this activity uses the URL and credentials of the public sandbox.

Part 1: Using Postman to Get a Network Host Inventory

Postman is an excellent tool for learning about an API before writing code for it. It is a good practice to visit the Swagger page for an endpoint to learn about the request requirements, and then try your request out in Postman. After verifying the requirements for accessing the API in Postman, and reviewing the JSON data that is returned, you can move on to coding the request in Python.

Step 1: Configure and send the Postman request for a host inventory.

a. Click the plus sign (+) to create a new tab.

b. Enter the following information:

  1) Request method: **GET**

  2) Endpoint URI: **https://{YOUR-APICEM}.cisco.com/api/v1/host**

  3) Headers:

     Content-Type: application/json

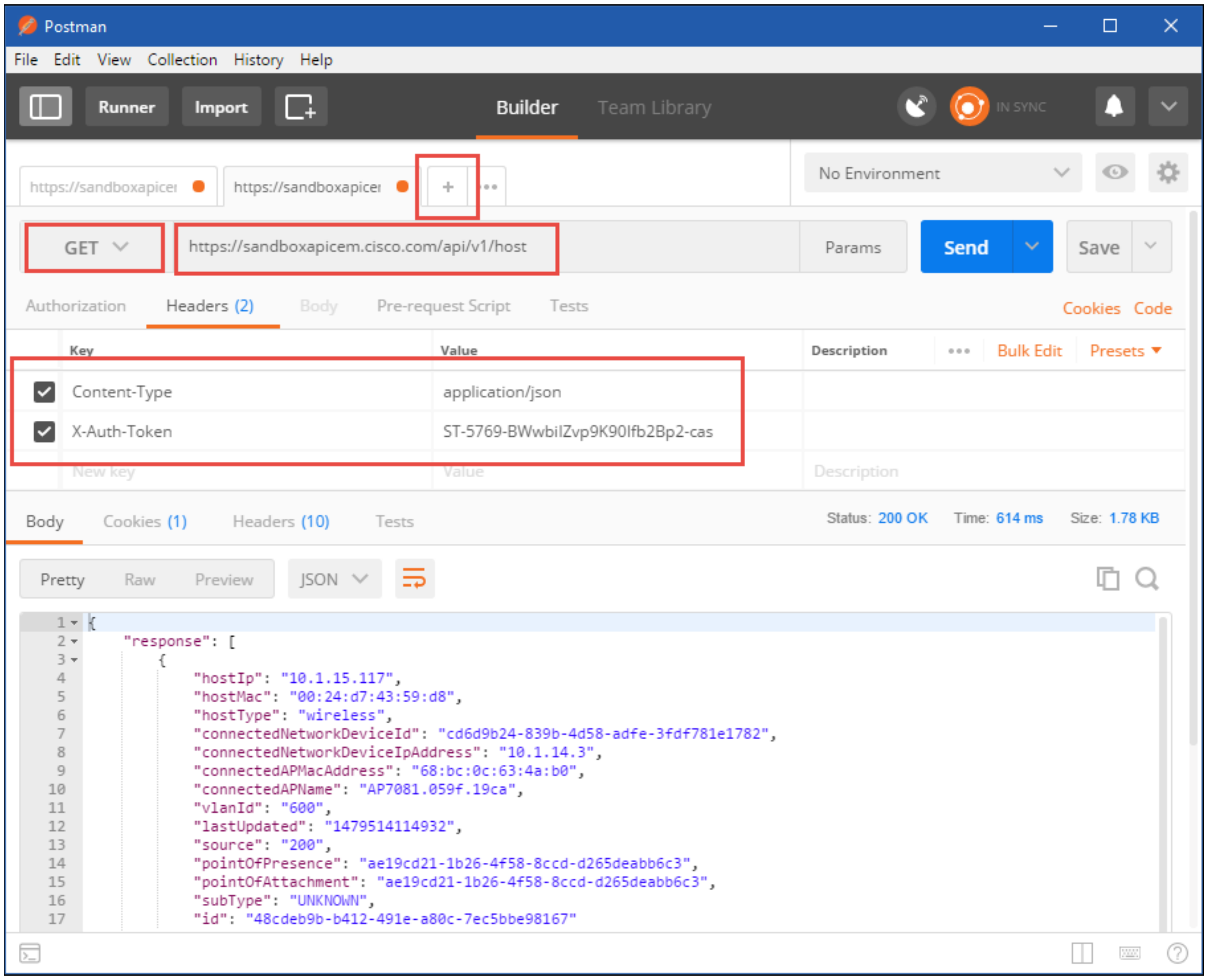
X-Auth-Token: <leave this blank for now>

c. Click the tab for the service ticket request you created in the previous lab and click **Send**.

d. Copy the value of the service ticket, without quotes, from the response JSON.

e. Return to the host inventory request tab and paste the value of the service ticket into the **Value** field for the **X-Auth-Token** key. See the figure below.

f. Click **Send**. If the request is successful, you will see the body section of response populate with JSON data that represents the host inventory, as shown below. If the response fails, look at the status value and try to determine where the error may be.



GET https://sandboxapicem.cisco.com/api/v1/host

Key	Value	Description
✓ Content-Type	application/json	
✓ X-Auth-Token	ST-5769-BWwbilZvp9K90lfb2Bp2-cas	

Status: 200 OK Time: 614 ms Size: 1.78 KB

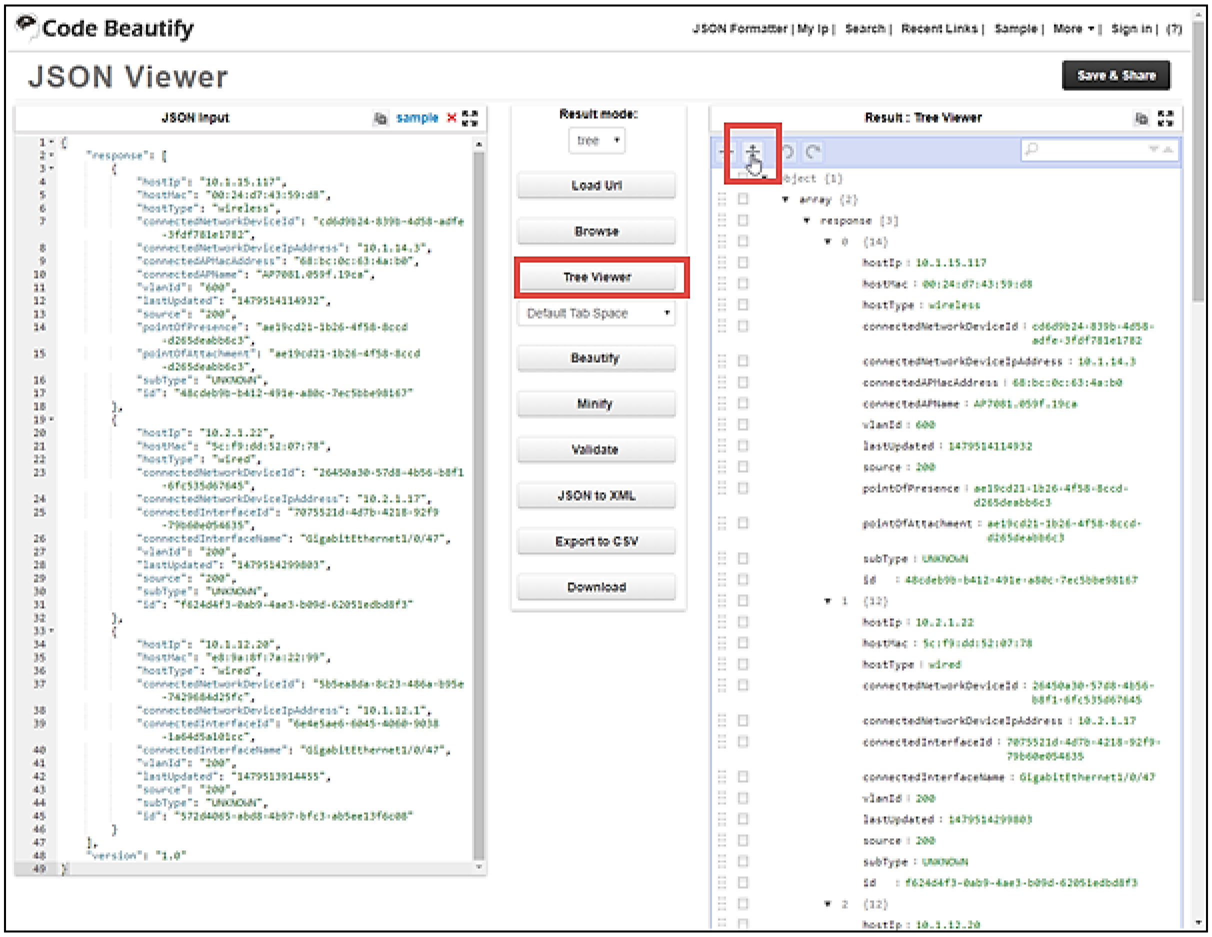
```
{
    "response": [
        {
            "hostIp": "10.1.15.117",
            "hostMac": "00:24:d7:43:59:d8",
            "hostType": "wireless",
            "connectedNetworkDeviceId": "cd6d9b24-839b-4d58-adfe-3fdf781e1782",
            "connectedNetworkDeviceIpAddress": "10.1.14.3",
            "connectedAPMacAddress": "68:bc:0c:63:4a:b0",
            "connectedAPName": "AP7081.059f.19ca",
            "vlanId": "600",
            "lastUpdated": "1479514114932",
            "source": "200",
            "pointOfPresence": "ae19cd21-1b26-4f58-8ccd-d265deabb6c3",
            "pointOfAttachment": "ae19cd21-1b26-4f58-8ccd-d265deabb6c3",
            "subType": "UNKNOWN",
            "id": "48cdeb9b-b412-491e-a80c-7ec5bbe98167"
```

Step 2: Use CodeBeautfiy.com to evaluate the response.

Code Beautify maintains a website for viewing code in a more human readable format. The JSON viewer URL is https://codebeautify.org/jsonviewer.

a. Copy the JSON from Postman and paste it into the left window in JSON Viewer.

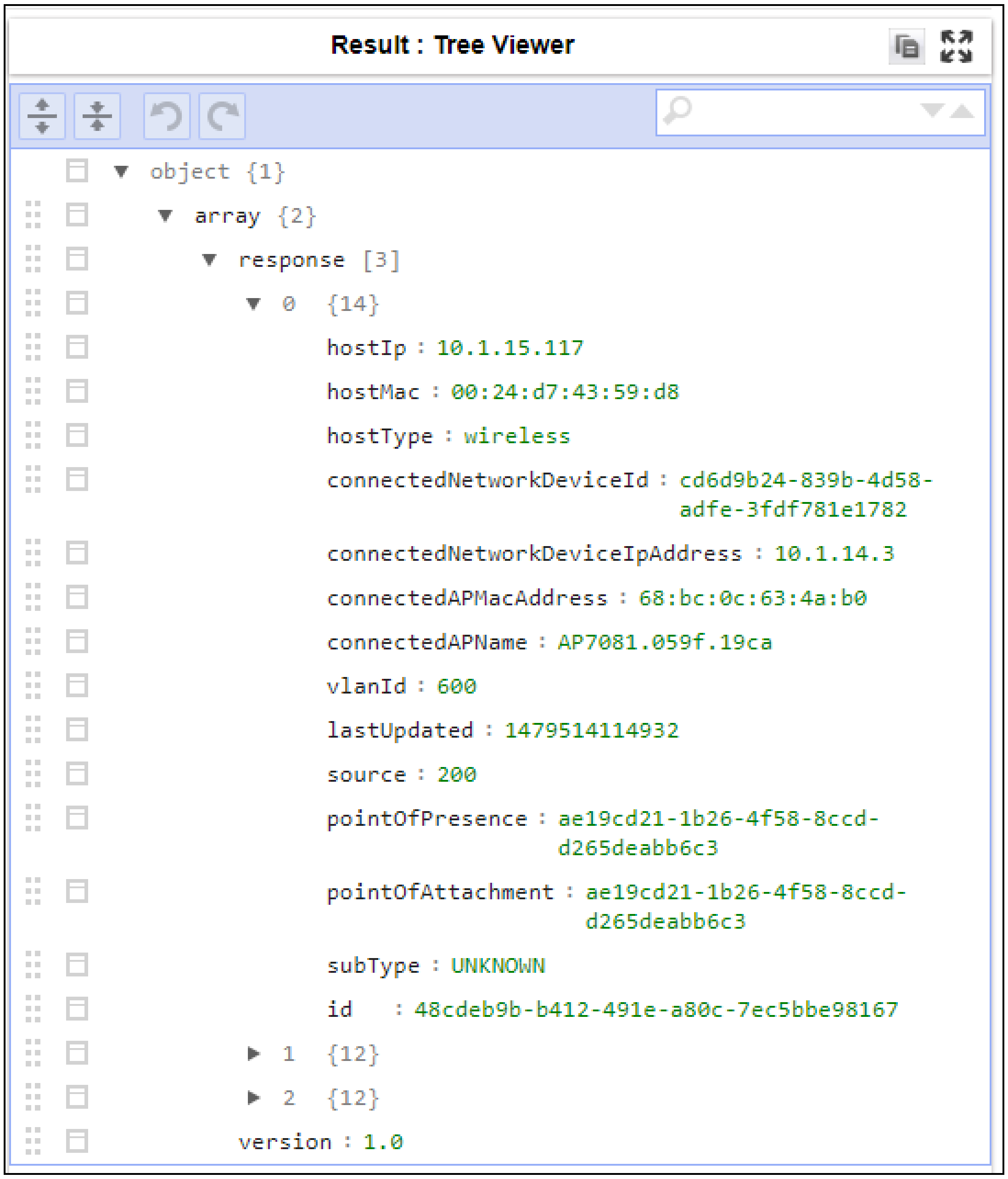
b. Click **Tree Viewer** to render the tree, as shown below:



```
{
    "response": [
        {
            "hostIp": "10.1.15.117",
            "hostMac": "00:24:d7:43:59:d8",
            "hostType": "wireless",
            "connectedNetworkDeviceId": "cd6d9b24-839b-4d58-adfe-3fdf781e1782",
            "connectedNetworkDeviceIpAddress": "10.1.14.3",
            "connectedAPMacAddress": "68:bc:0c:63:4a:b0",
            "connectedAPName": "AP7081.059f.19ca",
            "vlanId": "600",
            "lastUpdated": "1479514114932",
            "source": "200",
            "pointOfPresence": "ae19cd21-1b26-4f58-8ccd-d265deabb6c3",
            "pointOfAttachment": "ae19cd21-1b26-4f58-8ccd-d265deabb6c3",
            "subType": "UNKNOWN",
            "id": "48cdeb9b-b412-491e-a80c-7ec5bbe98167"
        },
        {
            "hostIp": "10.2.1.22",
            "hostMac": "5c:f9:dd:52:07:78",
            "hostType": "wired",
            "connectedNetworkDeviceId": "26450a30-57d8-4b56-b8f1-6fc535d67645",
            "connectedNetworkDeviceIpAddress": "10.2.1.17",
            "connectedInterfaceId": "7075521d-4d7b-4218-92f9-79b60e054635",
            "connectedInterfaceName": "GigabitEthernet1/0/47",
            "vlanId": "200",
            "lastUpdated": "1479514299803",
            "source": "200",
            "subType": "UNKNOWN",
            "id": "f624d4f3-0ab9-4ae3-b09d-62051edbd8f3"
        },
        {
            "hostIp": "10.1.12.20",
            "hostMac": "e8:9a:8f:7a:22:99",
            "hostType": "wired",
            "connectedNetworkDeviceId": "5b5ea8da-8c23-486a-b95e-7429684d25fc",
            "connectedNetworkDeviceIpAddress": "10.1.12.1",
            "connectedInterfaceId": "6e4e5ae6-6045-4060-9038-1a64d5a101cc",
            "connectedInterfaceName": "GigabitEthernet1/0/47",
            "vlanId": "200",
            "lastUpdated": "1479513914455",
            "source": "200",
            "subType": "UNKNOWN",
            "id": "572d4065-abd8-4b97-bfc3-ab5ee13f6c08"
        }
    ],
    "version": "1.0"
}
```

```
object {1}
  ▼ array {2}
    ▼ response [3]
      ▼ 0  {14}
            hostIp : 10.1.15.117
            hostMac : 00:24:d7:43:59:d8
            hostType : wireless
            connectedNetworkDeviceId : cd6d9b24-839b-4d58-adfe-3fdf781e1782
            connectedNetworkDeviceIpAddress : 10.1.14.3
            connectedAPMacAddress : 68:bc:0c:63:4a:b0
            connectedAPName : AP7081.059f.19ca
            vlanId : 600
            lastUpdated : 1479514114932
            source : 200
            pointOfPresence : ae19cd21-1b26-4f58-8ccd-d265deabb6c3
            pointOfAttachment : ae19cd21-1b26-4f58-8ccd-d265deabb6c3
            subType : UNKNOWN
            id    : 48cdeb9b-b412-491e-a80c-7ec5bbe98167
      ▼ 1  {12}
            hostIp : 10.2.1.22
            hostMac : 5c:f9:dd:52:07:78
            hostType : wired
            connectedNetworkDeviceId : 26450a30-57d8-4b56-b8f1-6fc535d67645
            connectedNetworkDeviceIpAddress : 10.2.1.17
            connectedInterfaceId : 7075521d-4d7b-4218-92f9-79b60e054635
            connectedInterfaceName : GigabitEthernet1/0/47
            vlanId : 200
            lastUpdated : 1479514299803
            source : 200
            subType : UNKNOWN
            id    : f624d4f3-0ab9-4ae3-b09d-62051edbd8f3
      ▼ 2  {12}
            hostIp : 10.1.12.20
```

c. Collapse all levels by clicking the second icon in the Result window, as shown above. Expand **object**, **array**, and **response**. The number next to the response key indicates how many entries there are. Expand the level marked **0**. Your Tree Viewer should look similar to the following:



```
▼ object {1}
    ▼ array {2}
        ▼ response [3]
            ▼ 0   {14}
                  hostIp : 10.1.15.117
                  hostMac : 00:24:d7:43:59:d8
                  hostType : wireless
                  connectedNetworkDeviceId : cd6d9b24-839b-4d58-
                                             adfe-3fdf781e1782
                  connectedNetworkDeviceIpAddress : 10.1.14.3
                  connectedAPMacAddress : 68:bc:0c:63:4a:b0
                  connectedAPName : AP7081.059f.19ca
                  vlanId : 600
                  lastUpdated : 1479514114932
                  source : 200
                  pointOfPresence : ae19cd21-1b26-4f58-8ccd-
                                    d265deabb6c3
                  pointOfAttachment : ae19cd21-1b26-4f58-8ccd-
                                      d265deabb6c3
                  subType : UNKNOWN
                  id    : 48cdeb9b-b412-491e-a80c-7ec5bbe98167
            ▶ 1   {12}
            ▶ 2   {12}
        version : 1.0
```

d. Look at the key/value pairs that are assigned to level 0.

What does level 0 represent? How about levels 1 and 2?

____

____

Next to level 0 in the example, the number 14 indicates that there are 14 keys associated with this entry. However, there are different numbers for the other hosts. Open each device and compare entries. Why is there a difference?

____

____

Part 2: Using Python to Get a Network Host Inventory

In this part of the lab, you will create a Python program to get the same network host inventory you retrieved using Postman in Part 1. You will then create a function from your program and add it to your **my_apic_em_functions.py** file.

Step 1: Setup the code environment.

a. Open IDLE and click **File > New File**. Save the new file as **print_hosts.py**. Make sure it is saved in the same location with your other files.

b. Import the **requests** and **json**, modules.

c. Import **tabulate** and **my_apic_em_functions** using **from… import *** statements. You will use the **tabulate** function to format and print a table. You will use your **get_ticket()** function in this lab to request a fresh service ticket. If necessary, import the **apic_em_functions_sol** module if a working version of **my_apic_em_functions** is unavailable.

```
import requests
import json
from tabulate import *
from my_apic_em_functions import *
```

Step 2: Build the request components.

a. Create the variable **api_url** and assign a string containing the URI of the APIC-EM **/host** endpoint that you are using for this workshop. Refer to the work you did for the last lab if you need help.

```
api_url = "https://{YOUR-APICEM}.cisco.com/api/v1/host"
```

b. Create the variable **ticket** and assign to it the value returned by the **get_ticket()** function.

```
ticket = get_ticket()
```

c. Create the **headers** dictionary and assign it to the **headers** variable. The header needs to include the content type and authentication token as it did in Postman. The ticket variable created in the last step is used as the **X-Auth-Token** value. This is created as a two-key dictionary. Be careful with the punctuation.

```
headers = {
    "content-type": "application/json",
    "X-Auth-Token": ticket
}
```

Step 3: Make the request and handle errors.

a. Create a variable named **resp**, and assign to it the results of the API request. The **requests.get()** method is used to make the request and supply it with the URI and headers variables created above as arguments as well as a couple of necessary settings.

```
resp = requests.get(api_url, headers=headers, verify=False)
```

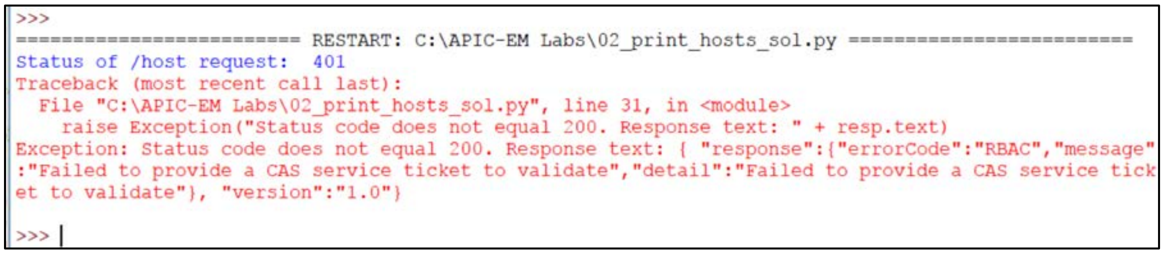
b. Next, print the status of the request to display the status of the request. Because **resp** is a **requests** object, it has a property called **status_code.** Print a string with some explanatory text followed by the status code.

```
print("Status of /host request: ", resp.status_code)
```

c. If the API response status code of the request is different from code 200 (that indicates OK), the application should stop the execution because most probably an error has occurred in the API call. Verify with a simple **if** statement if the value of the resp.status_code equals 200. If not, raise an exception to stop the execution:

```
if resp.status_code != 200:
    raise Exception("Status code does not equal 200. Response text: " + resp.text)
```

This creates a custom error message that can help to identify the root cause of the error. As shown below.



```
>>>
========================= RESTART: C:\APIC-EM Labs\02_print_hosts_sol.py =========================
Status of /host request:  401
Traceback (most recent call last):
  File "C:\APIC-EM Labs\02_print_hosts_sol.py", line 31, in <module>
    raise Exception("Status code does not equal 200. Response text: " + resp.text)
Exception: Status code does not equal 200. Response text: { "response":{"errorCode":"RBAC","message"
:"Failed to provide a CAS service ticket to validate","detail":"Failed to provide a CAS service tick
et to validate"}, "version":"1.0"}

>>> |
```

This output indicates there is an authorization problem possibly because the service ticket is invalid or there is a problem with the **get_ticket()** function.

d. Convert the JSON response data into a Python dictionary format with the following statement:

```
response_json = resp.json()
```

The entire block of code should look like this:

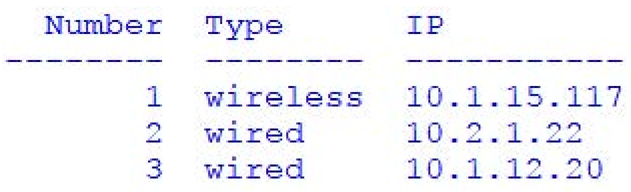
```
resp = requests.get(api_url, headers=headers, verify=False)
print("Status of /host request: ", resp.status_code)
if resp.status_code != 200:
    raise Exception("Status code does not equal 200. Response text: " + resp.text)
response_json = resp.json()
```

e. Save your script and run it. You should get output similar to the following. If not, troubleshoot your code for errors.

```
The service ticket number is: ST-5873-WBA3XaHjev0aJYeWPVk3-cas
Status of /host request: 200
>>>
```

Step 4: Parse and format the JSON response data.

Now you are ready to extract specific host information from the JSON response data. You will write the code to parse the JSON data and create a table similar to the following:



```
  Number  Type       IP
--------  ---------  -----------
       1  wireless   10.1.15.117
       2  wired      10.2.1.22
       3  wired      10.1.12.20
```

To do this, a **for:** loop is created. The **for:** loop will iterate through the list of hosts and extract the value for the two desired dictionary keys. They are the **hostType** and **hostIP** keys of the **response** dictionary. You will create a list variable called **host_list** that will contain the values for the Number, Type, and IP columns. Each host will be appended to the list as a separate line as the loop executes. Execution of loop stops when there are no more items under the **response** dictionary key.

The number of the host is not present in the current JSON data. You will create a separate variable, assign a value of 0 and then increment it as the loop repeats.

a. First initialize the two required variables. The first is a list variable called host_list. The second is an integer variable that will hold the value for the Number column in the table.

```
host_list = []
i = 0
```

b. Now create the **for:** loop. The loop will iterate over every item in the response key of the **response_json** variable. Each time that loop iterates, the **item** variable takes on the value of the JSON for that item in response, which corresponds to each host. The statement that does this is:

```
for item in response_json["response"]:
```

c. The next lines of code will execute as the **for:** loop runs. They should be indented. First, the ordinal number variable **i** is incremented through each loop:

```
i += 1
```

Then, append the values from the JSON to the **host_list** variable. The key to understanding this line of code is that the variable **item** takes on the value of the keys under the response key with every iteration. In essence, **item** holds all the information in the JSON for device 0 in the first iteration, device 1 in the next, etc. This means that **item** has the **hostType** and **hostIP** keys within it. The code appends the value for the ordinal variable **i**, the value for **hostType**, and the value for **hostIP** to the **host_list** variable. As the loop iterates it does this for each host.

```
host = [ i, item["hostType"], item["hostIp"] ]
host_list.append( host )
```

The complete **for:** loop should look like this:

```
for item in response_json["response"]:
    i+=1
    host = [
            i,
            item["hostType"],
            item["hostIp"]
            ]
    host_list.append( host )
```

d. Finally, use the **tabulate** function to print the table of hosts. The tabulate function will take as arguments the **host_list** variable, a list of headers for the columns that will be printed, and a setting for the table format.

```
table_header = ["Number", "Type", "IP"]
print( tabulate(host_list, table_header) )
```

e. Save your script and run it. You should get a table similar to the one shown at the beginning of this step. Investigate any errors that may occur. Look for mismatched paired symbols like (), [], and " ".

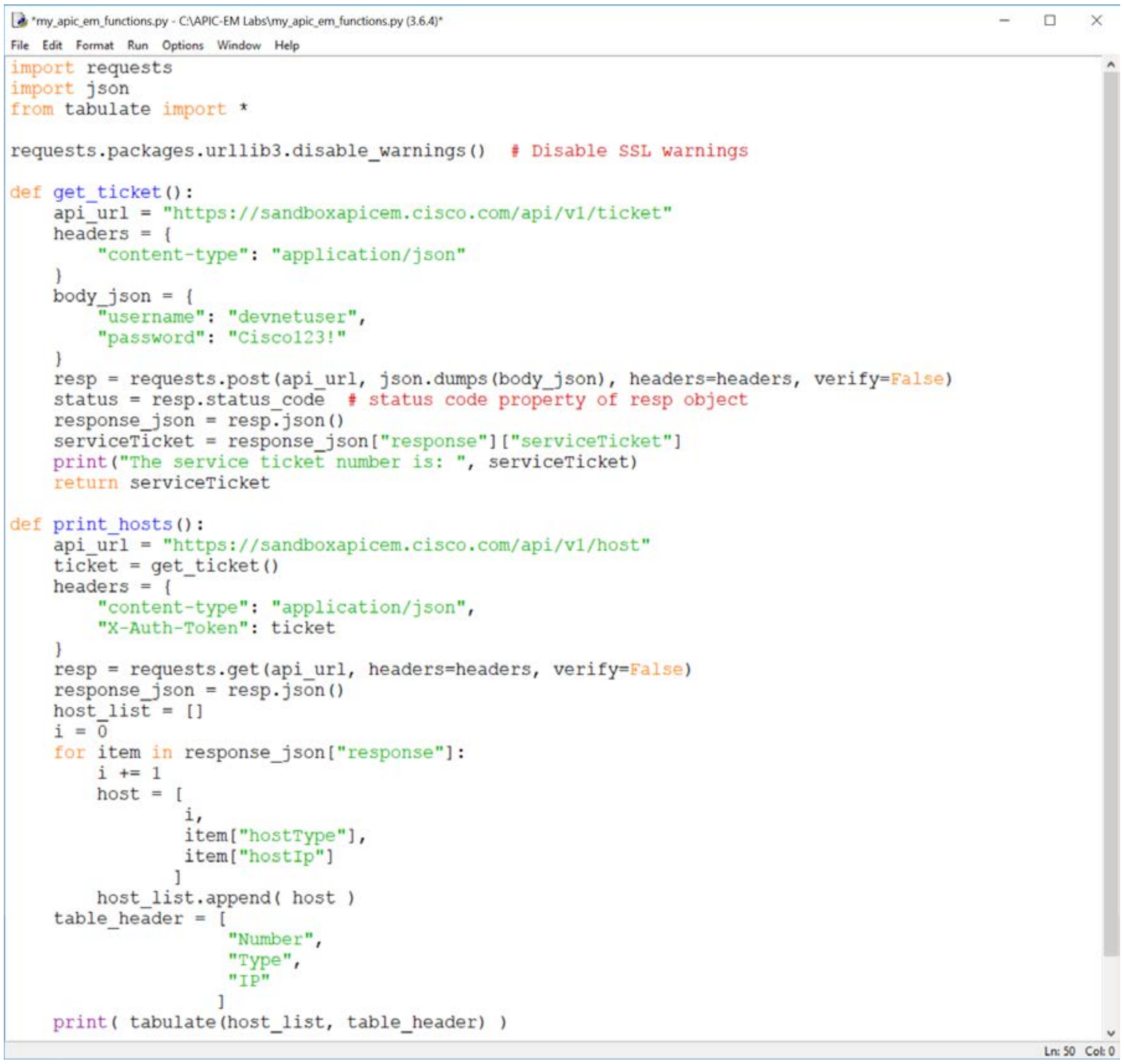
Step 5: Create the function for the host inventory request.

You will now add this code to the **my_apic_em_functions.py** file that contains your **get_ticket()** function.

a. The **my_apic_em_functions.py** should already have the import statements for **requests** and **json**. Copy and paste the **tabulate** import statement into the file below them.

b. Add a few blank lines below the **get_ticket()** function and define a new function called **print_hosts():**. Copy your code from the **api_url** variable through the print statement and paste it below **print_hosts():**.

c. Select all of the lines below the function definition statement and select **Indent Region** from the IDLE format menu, or press the TAB key. Everything should be indented an additional four spaces. This function does not require a **return** statement. Your functions file should look like the following. Make sure your indention is correct for each line of code:


my_apic_em_functions.py - C:\APIC-EM Labs\my_apic_em_functions.py (3.6.4)
File Edit Format Run Options Window Help
Ln: 50 Col: 0

```
import requests
import json
from tabulate import *

requests.packages.urllib3.disable_warnings()  # Disable SSL warnings

def get_ticket():
    api_url = "https://sandboxapicem.cisco.com/api/v1/ticket"
    headers = {
        "content-type": "application/json"
    }
    body_json = {
        "username": "devnetuser",
        "password": "Cisco123!"
    }
    resp = requests.post(api_url, json.dumps(body_json), headers=headers, verify=False)
    status = resp.status_code  # status code property of resp object
    response_json = resp.json()
    serviceTicket = response_json["response"]["serviceTicket"]
    print("The service ticket number is: ", serviceTicket)
    return serviceTicket

def print_hosts():
    api_url = "https://sandboxapicem.cisco.com/api/v1/host"
    ticket = get_ticket()
    headers = {
        "content-type": "application/json",
        "X-Auth-Token": ticket
    }
    resp = requests.get(api_url, headers=headers, verify=False)
    response_json = resp.json()
    host_list = []
    i = 0
    for item in response_json["response"]:
        i += 1
        host = [
                i,
                item["hostType"],
                item["hostIp"]
               ]
        host_list.append( host )
    table_header = [
                    "Number",
                    "Type",
                    "IP"
                   ]
    print( tabulate(host_list, table_header) )
```

d. Save and run the functions file. In the IDLE Shell, type **print hosts()** at the prompt. The function should run just as your program did.