

## Lab – RESTCONF with Python

### Import modules and disable SSL warnings.

- In IDLE, click **File > New File** to open IDLE Editor.
- Save the file as **lab 2.5.py**.
- Enter the following commands to import the modules and disable SSL certificate warnings:

```
import json
import requests
requests.packages.urllib3.disable_warnings()
```

The **json** module includes methods convert JSON data to Python objects and vice versa. The **requests** module has methods that will let us send REST requests to a URI.

### Step 1: Build the request components.

Create a string variable to hold the API endpoint URI and two dictionaries, one for the request header and one for the body JSON. These are the same tasks you completed in the Postman application.

- Create a variable named **api\_url** and assign the URL (adjust the IP address to match the router's current address).

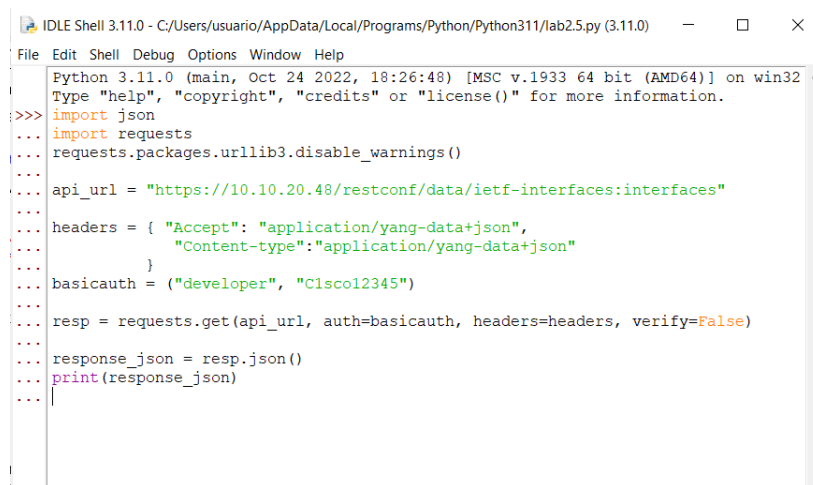
```
api_url = "https://192.168.56.101/restconf/data/ietf-interfaces:interfaces"
```

- Create a dictionary variable named **headers** that has keys for **Accept** and **Content-type** and assign the keys the value **application/yang-data+json**.

```
headers = { "Accept": "application/yang-data+json",
            "Content-type": "application/yang-data+json"
          }
```

- Create a Python tuple variable named **basicauth** that has two keys needed for authentication, **username** and **password**.

```
basicauth = ("cisco", "cisco123!")
```



```
IDLE Shell 3.11.0 - C:/Users/usuario/AppData/Local/Programs/Python/Python311/lab2.5.py (3.11.0)
File Edit Shell Debug Options Window Help
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import json
... import requests
... requests.packages.urllib3.disable_warnings()
...
... api_url = "https://10.10.20.48/restconf/data/ietf-interfaces:interfaces"
...
... headers = { "Accept": "application/yang-data+json",
...             "Content-type": "application/yang-data+json"
...           }
... basicauth = ("developer", "C1sco12345")
...
... resp = requests.get(api_url, auth=basicauth, headers=headers, verify=False)
...
... response_json = resp.json()
... print(response_json)
...
... |
```

## Step 2: Send the request.



```

IDLE Shell 3.11.0
File Edit Shell Debug Options Window Help
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
== RESTART: C:/Users/usuario/AppData/Local/Programs/Python/Python311/lab2.5.py =
Traceback (most recent call last):
  File "C:/Users/usuario/AppData/Local/Programs/Python/Python311/lab2.5.py", line 2, in <module>
    import requests
ModuleNotFoundError: No module named 'requests'
>>> == RESTART: C:/Users/usuario/AppData/Local/Programs/Python/Python311/lab2.5.py =
{'ietf-interfaces:interfaces': {'interface': [{'name': 'GigabitEthernet1', 'description': 'MANAGEMENT INTERFACE - DON'T TOUCH ME', 'type': 'iana-if-type:ethernetCsmacd', 'enabled': True, 'ietf-ip:ipv4': {'address': [{'ip': '10.10.20.48', 'netmask': '255.255.255.0'}]}, 'ietf-ip:ipv6': {}}, {'name': 'GigabitEthernet2', 'description': 'Network Interface', 'type': 'iana-if-type:ethernetCsmacd', 'enabled': False, 'ietf-ip:ipv4': {}, 'ietf-ip:ipv6': {}}, {'name': 'GigabitEthernet3', 'description': 'Network Interface', 'type': 'iana-if-type:ethernetCsmacd', 'enabled': False, 'ietf-ip:ipv4': {}, 'ietf-ip:ipv6': {}}, {'name': 'Loopback1', 'description': 'LABORATORIO 2.2', 'type': 'iana-if-type:softwareLoopback', 'enabled': True, 'ietf-ip:ipv4': {'address': [{'ip': '1.1.1.1', 'netmask': '255.255.255.0'}]}, 'ietf-ip:ipv6': {}}, {'name': 'Loopback99', 'description': 'LABORATORIO 2.2 99', 'type': 'iana-if-type:softwareLoopback', 'enabled': True, 'ietf-ip:ipv4': {'address': [{'ip': '99.99.99.99', 'netmask': '255.255.255.0'}]}, 'ietf-ip:ipv6': {}}]}}
>>>

```

```

10.10.20.48 - PuTTY
login as: developer
Keyboard-interactive authentication prompts from server:
Password:
End of keyboard-interactive prompts from server

Welcome to the DevNet Sandbox for CSR1000v and IOS XE

The following programmability features are already enabled:
- NETCONF
- RESTCONF

Thanks for stopping by.

csr1000v-1#
csr1000v-1#sh ip int brief
Interface          IP-Address      OK? Method Status        Protocol
GigabitEthernet1   10.10.20.48     YES NVRAM  up             up
GigabitEthernet2   unassigned      YES NVRAM  administrativ down down
GigabitEthernet3   unassigned      YES NVRAM  administrativ down down
Loopback1          1.1.1.1         YES manual up             up
csr1000v-1#

```

## Conclusiones

Lo que se realizó en esta practica fue basicante utilizar distintas variables que nos ayudarían obtener resultados con los parametros de `request.get()`.

Lo cual nos dice que este metodo es una libreria que hace peticiones en http que facilita enormemente el trabajo con peticiones HTTP. Antes o después, en algún proyecto, es posible que tengas que hacer peticiones web, ya sea para consumir un API, extraer información de una página o enviar el contenido de un formulario de manera automatizada. Si es así, Python requests es tu gran aliada.

Investigando más a fondo y junto con la información del laboratorio encontré algunos códigos de respuesta que `request.get` envía si realizamos una solicitud.

Código Descripción Humana

201 Creado OK, respuesta correcta frente a un POST.

204 Modificado OK, respuesta correcta frente a un PUT

400 Error en la petición, suele ocurrir frente a URIs mal armadas o con campos incorrectos en el cuerpo del mensaje.

401 Error en las credenciales.

404 No se encuentra el recurso. Posible problema en la URI

409 Conflicto en la creación. En un POST, significa que ya existe el recurso (se debería modificar con PUT).

`resp.status_code=` El código de estado HTTP en la respuesta de solicitud de API

Por otra parte podemos hacer lo mismo con Postman o con una terminal de Python, para obtener así un resultado añadiendo algún ciclo que nos confirme el resultado obtenido ya sea bueno o malo