

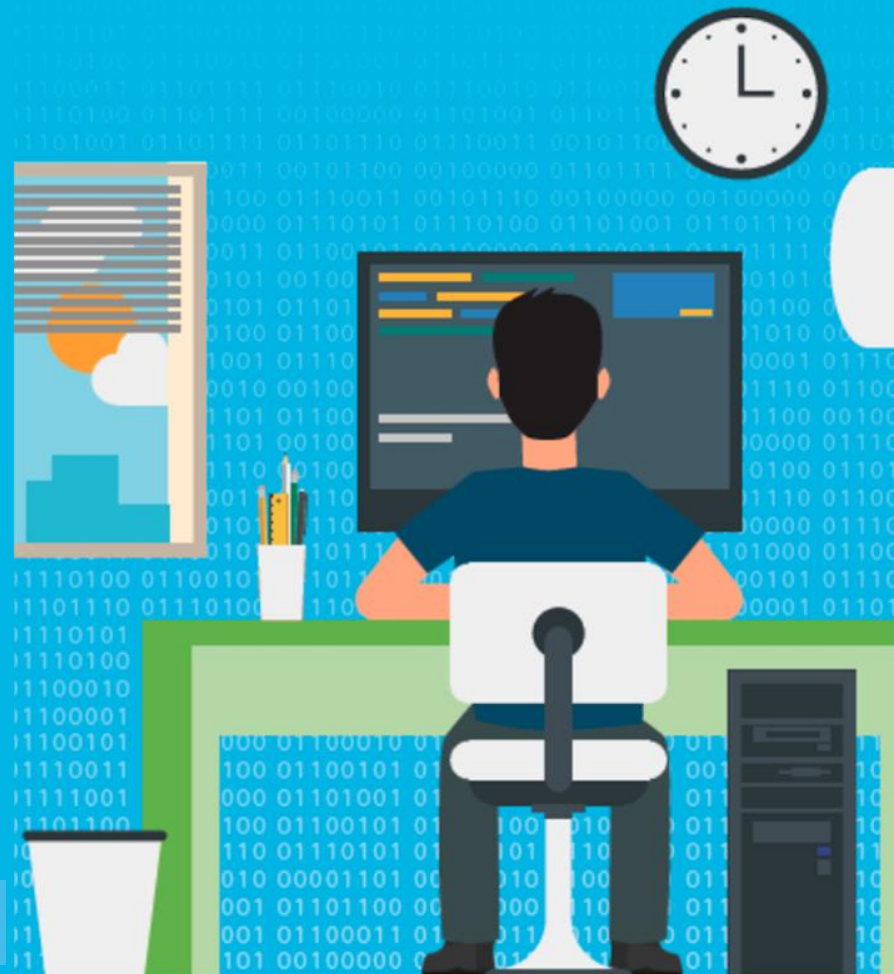
Python & JSON Workshop

Presentation v1.64

2021 February IPD Week

#NetAcadIPD

Yvan Rooseleer, Belgian IT Academy Support Center BiASC
Odisee University Of Applied Sciences, Brussels <yvan@biasc.be>





Yvan Rooseleer

Brussels, Belgium

Senior Lecturer

Odisee, Brussels

- University of Applied Sciences

BiASC Country Manager

- Instructor Trainer

> 10.000 h

Software Development

Database Development

> 10.000 h

Database Administration

IT Infrastructure Management

> 10.000 h

Network & Security Management

Problem-based challenges



Agenda

1

Practical examples with Python & JSON

Response Data in JSON

RESTCONF

Cisco DNA Center

Webex

Ansible

Docker

2

Processing data from Excel

Transforming data into a JSON tree structure

```
{
  "ietf-interfaces:interfaces": {
    "interface": [{
      "name": "GigabitEthernet1",
      "description": "VBox",
      "type": "iana-if-type:ethernetCsmacd",
      "enabled": true,
      "ietf-ip:ipv4": {
        "address": [{
          "ip": "192.168.56.101",
          "netmask": "255.255.255.0"
        }]
      },
      "ietf-ip:ipv6": {}
    },
    {
      "name": "Loopback9",
      "description": "Lo9",
      "type": "iana-if-type:softwareLoopback",
      "enabled": true,
      "ietf-ip:ipv4": {
        "address": [{
          "ip": "10.9.9.9",
          "netmask": "255.255.255.0"
        }]
      },
      "ietf-ip:ipv6": {}
    }
  ]
}
```

Parts of the Workshop

Part 1: DevNet Associate: Filtering JSON Data

Description: This part covers some aspects of the DevNet Associate course. Join to learn how to interpret JSON data coming from Webex Teams API, DNAC API, Docker inspect, Ansible.

Part 2: DevNet Associate: Generating JSON Data

Description: This part covers some aspects of the DevNet Associate course. Join to learn how to generate JSON data for Webex Teams API, IP Devices and Network Services from a source in the Excel format.

The examples in this workshop are based on the Emerging Technology Workshops and on the recent DevNet Associate Course from Cisco Networking Academy.

Useful Coding Skills for NetAcad Instructors

If you are teaching the DevNet Associate Course or the Emerging Technology Workshops, the following Python coding skills are relevant:

- ✓ Constructing API Calls using Python
 - URI, URL
 - Authentication & Authorization
- ✓ Managing JSON Exchange Data using Python *= This Workshop*
 - Response data
 - Data Types
 - Data Conversion
- ✓ Using existing Python Code and Libraries
 - Coding from scratch is not a requirement
 - This workshop focuses on JSON Data & Python dict and list

Python & JSON Workshop Part 1

Part 1: DevNet Associate: Filtering JSON Data

Description: This part covers some aspects of the DevNet Associate course. Join to learn how to interpret JSON data coming from Webex Teams API, DNAC API, Docker inspect, Ansible.

Part 2: DevNet Associate: Generating JSON Data

Description: This part covers some aspects of the DevNet Associate course. Join to learn how to generate JSON data for Webex Teams API, IP Devices and Network Services from a source in the Excel format.

Problem Statement

Response Data in JSON Format

- Machine readable
- Hierarchical Tree structure
- **Overwhelming** for humans

```
[
  {
    "Name": "bridge",
    "Id": "566a72fc961157e2e71cc257fc2132beebc491a712967ef42bddcab70cbdbb23",
    "Created": "2020-12-09T17:51:15.816558163Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
        {
          "Subnet": "172.17.0.0/16",
          "Gateway": "172.17.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": [
      {
        "4e99a64e10dfcf6608a1d47f4349676c745bf234cebd52826d786db9a3be2811": {
          "Name": "samplerunning",
          "EndpointID": "22bbd3fa7e76635c3172446813fe5104537c8f69c6c23474272b379dede44fe7",
          "MacAddress": "02:42:ac:11:00:03",
          "IPv4Address": "172.17.0.3/16",
          "IPv6Address": ""
        }
      ]
    },
    "Options": {
      "com.docker.network.bridge.default_bridge": "true",
      "com.docker.network.bridge.enable_icc": "true",
      "com.docker.network.bridge.enable_ip_masquerade": "true",
      "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
      "com.docker.network.bridge.name": "docker0",
      "com.docker.network.driver.mtu": "1500"
    },
    "Labels": {}
  }
]
```

Example 1: JSON RESPONSE DATA

RESTCONF URL Example/ Virtual Router

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

Authentication header is needed

Raw Response (serialized, machine readable)

```
{"ietf-interfaces:interfaces": {"interface": [{"name": "GigabitEthernet1", "description": "VBox", "type": "iana-if-type:ethernetCsmacd", "enabled": true, "ietf-ip:ipv4": {"address": [{"ip": "192.168.56.101", "netmask": "255.255.255.0"}]}, "ietf-ip:ipv6": {}}, {"name": "Loopback9", "description": "999", "type": "iana-if-type:softwareLoopback", "enabled": true, "ietf-ip:ipv4": {"address": [{"ip": "10.9.9.9", "netmask": "255.255.255.0"}]}, "ietf-ip:ipv6": {}}]}
```


Example 2: JSON RESPONSE DATA

DNA CENTER CLIENT HEALTH REQUEST

```
import requests
import json
```

```
# f-string is used for string formatting of request url
# Authentication header is needed
```

```
get_resp = requests.get(f"https://{host}/dna/intent/api/v1/client-health",
                        headers=headers, params=params)
```

RAW RESPONSE DATA FRAGMENT

```
{'response': [{'siteId': 'global', 'scoreDetail': [{'scoreCategory': {'scoreCategory': 'CLIENT_TYPE', 'value': 'ALL'}, 'scoreValue': 29, 'clientCount': 82, 'clientUniqueCount': 82, 'starttime': 1611071700000, 'endtime': 1611072000000}, {'scoreCategory': {'scoreCategory': 'CLIENT_TYPE', 'value': 'WIRED'}, 'scoreValue': 100, 'clientCount': 2, 'clientUniqueCount': 2, 'starttime': 1611071700000, 'endtime': 1611072000000, 'scoreList': [{'scoreCategory': {'scoreCategory': 'SCORE_TYPE', 'value': 'POOR'}, 'scoreValue': -1, 'clientCount': 0, 'clientUniqueCount': 0, 'starttime': 1611071700000, 'endtime': 1611072000000}, ...
```

Example 3: JSON RESPONSE DATA

Partial Output (with some response data highlighted)

```

{"id":"Y21zY29zcGFya...N1yZQ5NjI5MGY","emails":["Yvan.rooseleer@biasc.be"],"sipAddresses":[{"type":
"personal-room","value":"838744612@biasc.webex.com","primary":false},{type:"personal-
room","value":"yvan.rooseleer@biasc.webex.com","primary":false},{type:"cloud-
calling","value":"Yvan.rooseleer@biasc.calls.webex.com","primary":true}],displayName:"YvanRoosele
er","nickName":"Yvan","firstName":"Yvan","lastName":"Rooseleer","avatar":"https://avatar-prod-us-
east-2.webexcontent.com/Avtr~V1~e4d4112d-2548-4a47-810e-04fe64-a79b-49c5-823a-
92cec494bb9c1d1de6~1600","orgId":"Y21zY29zcGFyazovL3VzL09SR0FOSctODEwZS0wNGZlNDVlYTE4MWY","roles":["
Y21zY29zcGFya9hZG1pbg"],"licenses":["Y21tQTNDExZS1hYThkLTA1MDI3N2Y3Zjd1OQ","Y21zY29zcGFyazovL3VzL0
xJQ0VOY53ZWJleC5jb20","Y21zY29zcGFyazovL3VzL0xJQ0VOUXYmJlZTU2LWQwZmItNGFiNy1hMTYyLTlmNjQ2OGIyYmU5ZA
","Y21zYVVOU0UvZTRkNDExMmQtMjU0OC00MWBjNDI4MmQ5NmY5NA","Y21zYTJhNTNfZmlhc2Mud2ViZXguY29t","Y21zYZWEx
ODFmOkZTU18xYjcyOGZmOS03ZGU4LTRjYjctOTU0MC0yOTMyMGI1YTQyY2I","Y21zY29zUtMDRmZTZlZWExODFmOkZUTV9mNWZ
kZTM1Zi00NzA0LTQ2MGEtODEwZi00YzVkMzUyNDF1Njk","Y21zY29zcExODFmOkNHXzVkJyYjcwNjYyLWNmYTItNGFjZC04MZRlL
TgwYjNiNWVkJjNlZA","Y21zY29zcGFyazovL3jQ3MzgyOC1hOTgwLTQ3MmYtODE5ZC02YjZjY2UwOGU5MmI","Y21zY29zcGFy
azovLmOZZNU185ZWZhNzgxNC0zMzEzLTQ2NGYtOTY0Mi0wMjM5ODc1YmM5Zjg","Y21zY29zcGFyazovLNDExMmQtMjU0OGE3NS
ZhNDNmLTBkYmJhMjIyNzg3Zl9iaWFzYy53ZWJleC5jb20","Y21MwMjFkLTgwYjctNDFiYi1lZThhLWM0YjFiZjcyNTE4YV9iaW
FzYy53ZWJleC5jZ20","Y21zY29zcGFyazovZ3V0YTQ3LTgxMGUtMDRmZTQlZWExODFmOk1kYmFiMDcxYmY0NA","Y21zY29zcG
FyazovL3VzL0xJQ0VOUXZmZDU0MjM5ODc1YmM5Zjg","Y21zY29zcGFyazovL3VzL0xJQ0VOUXZmZDU0MjM5ODc1YmM5Zjg"],created:"2016-12-
23T08:38:22.877Z","lastModified":"2021-01-26T17:55:07.662Z","lastActivity":"2021-01-
26T17:54:24.481Z","status":"active","invitePending":false,"loginEnabled":true,"type":"person","trai
nSiteNames":["biasc.webex.com"]}

```

Example 4: JSON RESPONSE DATA

```
$ ansible webserver -m gather_facts --tree ./tmp_facts
```

URL for the complete response file:

Partial Output (with some response data highlighted)

```
{  
  "ansible_facts": {  
    "ansible_all_ipv4_addresses": ["192.0.2.1", "192.0.2.2", "192.0.2.3", "192.0.2.4", "192.0.2.5", "10.0.2.15", "172.17.0.1"],  
    "ansible_all_ipv6_addresses": ["fe80::9002:c8ff:fee8:bb09", "fe80::3c67:a5ff:fe17:e4cf", "fe80::a00:27ff:fee9:3de6", "fe80::42:3ff:fef6:9477"],  
    "ansible_apparmor": {"status": "enabled"},  
    "ansible_architecture": "x86_64",  
    "ansible_bios_date": "12/01/2006",  
    "ansible_bios_version": "VirtualBox",  
    "ansible_cmdline": {"BOOT_IMAGE": "/boot/vmlinuz-5.4.0-37-generic", "quiet": true, "ro": true, "root": "UUID=fb261367-cf98-4bce-b682-42b3de0a8ab9", "vga": "792", "zswap.enabled": "1"},  
    "ansible_date_time": {"date": "2021-01-20", "day": "20", "epoch": "1611160850", "hour": "16", "iso8601": "2021-01-20T16:40:50Z", "iso8601_basic": "20210120T164050181658", "iso8601_basic_short": "20210120T164050", "iso8601_micro": "2021-01-20T16:40:50.181774Z", "minute": "40", "month": "01", "second": "50", "time": "16:40:50", "tz": "UTC", "tz_offset": "+0000", "weekday": "Wednesday", "weekday_number": "3", "weeknumber": "03", "year": "2021"},  
    "ansible_default_ipv4": {"address": "10.0.2.15", "alias": "enp0s3", "broadcast": "10.0.2.255", "gateway": "10.0.2.2", "interface": "enp0s3", "macaddress": "08:00:27:e9:3d:e6", "mtu": 1500, "netmask": "255.255.255.0", "network": "10.0.2.0", "type": "ether"},  
    "ansible_default_ipv6": {},  
    "ansible_distribution": "Ubuntu",  
    "ansible_distribution_file_parsed": true,  
    "ansible_distribution_file_path": "/etc/os-release",  
    "ansible_distribution_file_variety": "Debian",  
    "ansible_distribution_major_version": "20",  
    "ansible_distribution_release": "focal",  
    "ansible_distribution_version": "20.04",  
    "ansible_dns": {"nameservers": ["127.0.0.53"], ... , "ansible_nodename": "labvm", "ansible_os_family": "Debian", "ansible_pkg_mgr": "apt", "ansible_proc_cmdline": {"BOOT_IMAGE": "/boot/vmlinuz-5.4.0-37-generic", "quiet": true, "ro": true, "root": "UUID=fb261367-cf98-4bce-b682-42b3de0a8ab9", "vga": "792", "zswap.enabled": "1"},  
    "ansible_processor": ["0", "GenuineIntel", "Intel(R) Core(TM) i7-7600U CPU @ 2.80GHz", "1", "GenuineIntel", "Intel(R) Core(TM) i7-7600U CPU @ 2.80GHz"],  
    "ansible_processor_cores": 2,  
    "ansible_processor_count": 1,  
    "ansible_processor_threads_per_core": 1,  
    "ansible_processor_vcpus": 2,  
    "ansible_product_name": "VirtualBox",  
    "ansible_product_serial": "NA",  
    "ansible_product_uid": "NA",  
    "ansible_product_version": "1.2",  
    "ansible_python": {"executable": "/usr/bin/python3", "has_sslcontext": true, "type": "cpython", "version": {"major": 3, "micro": 2, "minor": 8, "releaselevel": "final", "serial": 0}, "version_info": [3, 8, 2, "final", 0]},  
    "ansible_python_version": "3.8.2",  
    "ansible_real_group_id": 900,  
    "ansible_real_user_id": 900,  
    "ansible_selinux": {"status": "disabled"},  
    "ansible_selinux_python_present": true,  
    "ansible_service_mgr": "systemd",  
    "ansible_virtualization": {"address": "fe80::3c67:a5ff:fe17:e4cf", "prefix": "64", "scope": "link"}],  
    "macaddress": "3e:67:a5:17:e4:cf",  
    "mtu": 1500,  
    "promisc": true,  
    "speed": 10000,  
    "timestamping": ["tx_software", "rx_software", "software"],  
    "type": "ether"},  
    "ansible_virtualization_role": "guest",  
    "ansible_virtualization_type": "virtualbox",  
    "discovered_interpreter_python": "/usr/bin/python3",  
    "gather_subset": ["all"],  
    "module_setup": true,  
    "changed": false,  
    "deprecations": [],  
    "warnings": []  
  }  
}
```

Example 5: JSON RESPONSE DATA

\$ docker image inspect ubuntu

Output (with some response data highlighted)

```
[{"Id": "sha256:9140108b62dc87d9b278bb0d4fd6a3e44c2959646eb966b86531306faa81b09b", "RepoTags": ["ubuntu:latest"], "RepoDigests": ["ubuntu@sha256:bc2f7250f69267c9c6b66d7b6a81a54d3878bb85f1ebb5f951c896d13e6ba537"], "Parent": "", "Comment": "", "Created": "2020-09-25T22:34:30.295807036Z", "Container": "1046a5d685aef5c37d1829040ca8083b94e4c069ca4963f4b16aade2e077b06", "ContainerConfig": {"Hostname": "1046a5d685ae", "Domainname": "", "User": "", "AttachStdin": false, "AttachStdout": false, "AttachStderr": false, "Tty": false, "OpenStdin": false, "StdinOnce": false, "Env": ["PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"], "Cmd": ["/bin/sh", "-c", "#(nop) ", "CMD [\"/bin/bash\"]"], "ArgsEscaped": true, "Image": "sha256:4ff2090064e7e38688bce713d50f3202d227b3c89feceal434271c912ccd47e0", "Volumes": null, "WorkingDir": "", "Entrypoint": null, "OnBuild": null, "Labels": {}}, {"DockerVersion": "18.09.7", "Author": "", "Config": {"Hostname": "", "Domainname": "", "User": "", "AttachStdin": false, "AttachStdout": false, "AttachStderr": false, "Tty": false, "OpenStdin": false, "StdinOnce": false, "Env": ["PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"], "Cmd": ["/bin/bash"], "ArgsEscaped": true, "Image": "sha256:4ff2090064e7e38688bce713d50f3202d227b3c89feceal434271c912ccd47e0", "Volumes": null, "WorkingDir": "", "Entrypoint": null, "OnBuild": null, "Labels": null}, {"Architecture": "amd64", "Os": "linux", "Size": 72875723, "VirtualSize": 72875723, "GraphDriver": {"Data": {"LowerDir": "/var/lib/docker/overlay2/5d66f43ef0d92430a195045c4715ff3d49ee88ccb0bb7f6e69ccc5d60fde0ece /diff:/var/lib/docker/overlay2/0b75d53292ccb20230879e49c7fe6eeal57addb083d414ba3d0485959cd35086/diff", "MergedDir": "/var/lib/docker/overlay2/c3bab8487f590bfd66e53db0b1902ab5a8b8fbfalf02cfc3b4f0f9fa25158f2d/merged", "UpperDir": "/var/lib/docker/overlay2/c3bab8487f590bfd66e53db0b1902ab5a8b8fbfalf02cfc3b4f0f9fa25158f2d/diff", "WorkDir": "/var/lib/docker/overlay2/c3bab8487f590bfd66e53db0b1902ab5a8b8fbfalf02cfc3b4f0f9fa25158f2d/work"}, "Name": "overlay2"}, {"RootFS": {"Type": "layers", "Layers": ["sha256:d42a4fdf4b2ae8662ff2calb695eae571c652a62973c1beeb81a296a4f4263d92", "sha256:90ac32a0d9ab11e7745283f3051e990054616d631812ac63e324c1a36d2677f5", "sha256:782f5f011ddaf2a0bfd38cc2ccabd634095d6e35c8034302d788423f486bb177"]}, {"Metadata": {"LastTagTime": "0001-01-01T00:00:00Z"}}]}
```

Example 6: JSON RESPONSE DATA

```
device = []
try:
    # The request and response of "GET /network-device" API
    resp = get(api="network-device")
    status = resp.status_code
    # Get the json-encoded content from response
    response_json = resp.json()
    # all network-device detail is in "response"
    device = response_json["response"]

    # Try un-comment the following line to see what can we get
    print(json.dumps(device, indent=4))
except:
    print("Something wrong, cannot get network device information")
    sys.exit()

if status != 200:
    print(resp.text)
    sys.exit()

if device == []: # response is empty, no network-device is discovered.
    print("No network device found !")
    sys.exit()

device_list = []
# Now extract host name, ip and type to a list. Also add a sequential number in front
i=0
for item in device:
    i+=1
    device_list.append([i, item["hostname"], item["managementIpAddress"], item["type"], item["instanceUuid"]])

# We use tabulate module here to print a nice table format. You should use "pip" tool to install in your local machine
# For the simplicity we just copy the source code in working directory, didn't instal
# Not showing id to user, it's just a hex string
print(tabulate(device_list, headers=['number', 'hostname', 'ip', 'type'], tablefmt='rst'))
```

This for loop iterates through the `device` list

This line of code generates the output on the right. Each block - { } - represents one device.

```
{
  {
    "location": null,
    "type": "Cisco Nexus 3064 Switch",
    "snmpLocation": "",
    "role": "ACCESS",
    "collectionStatus": "Managed",
    "macAddress": "60:73:5c:e3:d5:7c",
    "platformId": "N3K-C3064PQ-10GE",
    "managementIpAddress": "10.10.30.16",
    "tunnelUpPort": null,
    "roleSource": "AUTO",
    "errorDescription": null,
    "inventoryStatusDetail": "<status><general code='SUCCESS'/></status>",
    "lastUpdated": "2016-11-12 02:36:43",
    "apManagerInterfaceIp": "",
    "upTime": "20 days, 23:26:23.62",
    "bootDateTime": "2016-10-22 03:10:36",
    "id": "a70f2449-be50-4454-8ff0-87e9fbcaefff",
    "hostname": "Nexus3k",
    "locationName": null,
    "series": "Data Center Switches",
    "interfaceCount": "67",
    "instanceUuid": "a70f2449-be50-4454-8ff0-87e9fbcaefff",
    "serialNumber": "FOC1637R0HS",
    "family": "Switches and Hubs",
    "snmpContact": "",
    "reachabilityStatus": "Reachable",
    "lastUpdateTime": "1478918203729",
    "softwareVersion": null,
    "memorySize": "3903836",
    "errorCode": null,
    "tagCount": "0",
    "lineCardId": "29471366-1b88-479b-a539-c439028cae06",
    "reachabilityFailureReason": "",
    "lineCardCount": "1"
  },
  {
    "location": null,
    "type": "Cisco Cloud Services Router 1000V",

```

Tips for NetAcad Instructors and Students

- Learn how to recognize **data type**

- `ietf_ipv4 = {'address': [{'ip': '192.168.56.101', 'netmask': '255.255.255.0'}]}`
 - `print(type(ietf_ipv4))`
 - Output: `<class 'dict'>`
- `multiple_addresses = [{'ip': '192.168.56.101', 'netmask': '255.255.255.0'}, {'ip': '192.0.2.1', 'netmask': '255.255.255.252'}]`
 - `print(type(multiple_addresses))`
 - Output: `<class 'list'>`

- Practice how to **select or filter** a data element

- `print(ietf_ipv4['address'][0]['ip'])` *### first ip address selected*
- Output: `192.168.56.101`
- Use the **keys()** function to determine the keys used in the response

Tips for NetAcad Instructors and Students (2)

- Practice how to **transform** data structures if necessary
 - `json.dumps`
 - `json.loads`
- Only **hands-on practice** helps to grasp the rules and procedures
 - Experiment with the code

Don't reinvent the wheel, don't program from scratch

- Import the necessary libraries in your Python scripts
 - `import requests`
 - `import json`
- Learn from example scripts

Data Types & Data Conversion

The following elements are necessary to grasp for this workshop:

data type

- str, int, boolean, dict, list and many more ...
- JSON = dict + list

dictionary: keys and values

- `{'hostname': 'CSR1kv'}`
- `{'ip address': '192.168.56.100'}`

list

- `['name', 'description', 'type', 'enabled', 'ietf-ip:ipv4', 'ietf-ip:ipv6']`

import json

- **Convert to dict:** `json.loads`
- **Convert to json** (serialization): `json.dumps`

Example A - Managing IP Addresses with Python

Python Data Structures for IP Addresses and Subnet Masks

Data Structures

```
single_address      = {'ip': '192.168.56.101', 'netmask': '255.255.255.0'}
multiple_addresses = [{'ip': '192.168.56.101', 'netmask': '255.255.255.0'},
                       {'ip': '192.0.2.1', 'netmask': '255.255.255.252'}]
ietf_ipv4           = {'address': [{'ip': '192.168.56.101', 'netmask': '255.255.255.0'}]}
```

Recognize Data Type & Select IP Address

```
print(type(single_address))           => <class 'dict'>
-
print(type(multiple_addresses))       => <class 'list'>
--
print(type(ietf_ipv4))                => <class 'dict'>
---
print(ietf_ipv4['address'][0]['ip'])   => 192.168.56.101
```

Example B - Managing IP Prefixes with Python

Python Data Structure for Network Prefixes and Subnet Masks (*fragment*)

```
prefix_subnet_masks = {  
    '/24': '255.255.255.0',  
    '/25': '255.255.255.128',  
    '/26': '255.255.255.192'  
}
```

example 1 -- using square brackets to select a key

```
subnet_mask_1 = prefix_subnet_masks['/24']
```

Output: 255.255.255.0

example 2 -- using get() function to select a key

```
subnet_mask_2 = prefix_subnet_masks.get('/26')
```

Output: 255.255.255.192

Exercise C - Managing IP Subnets with Python

Python Data Structure for Network Subnet Masks and Prefixes (*fragment*)

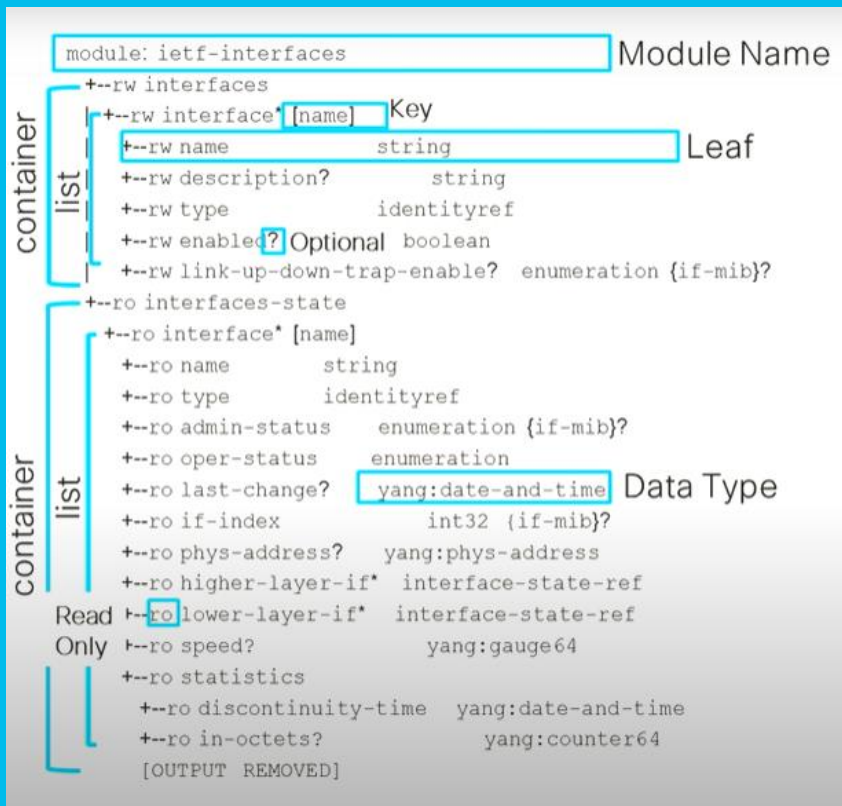
```
netmask_prefixes = {  
    '255.255.255.252': '/30'  
    , '255.255.255.248': '/29'  
    , '255.255.255.240': '/28'  
    , '255.255.255.224': '/27'  
    , '255.255.255.192': '/26'  
    , '255.255.255.128': '/25'  
    , '255.255.255.0'  : '/24'  
    , '255.255.254.0'  : '/23'  
}
```

Adapt to code fragment below to produce the output given

```
Prefix_1 = netmask_prefixes["your answer"]
```

Output: /27

YANG Data Model => RESTCONF JSON Example



```
{
  "ietf-interfaces:interfaces": {
    "interface": [{
      "name": "GigabitEthernet1",
      "description": "VBox",
      "type": "iana-if-type:ethernetCsmacd",
      "enabled": true,
      "ietf-ip:ipv4": {
        "address": [{
          "ip": "192.168.56.101",
          "netmask": "255.255.255.0"
        }]
      },
      "ietf-ip:ipv6": {}
    }]
  }
}
```

© 2020, Cisco and/or its affiliates. All rights reserved.

RESTCONF Response Data from CSR1000v

RESTCONF URL Example

URL: "https://192.168.56.101/restconf/data/ietf-interfaces:interfaces"

Authentication header needed

Raw Response (serialized, machine readable)

```
{"ietf-interfaces:interfaces": {"interface": [{"name": "GigabitEthernet1", "description": "VBox", "type": "iana-if-type:ethernetCsmacd", "enabled": true, "ietf-ip:ipv4": {"address": [{"ip": "192.168.56.101", "netmask": "255.255.255.0"}]}, "ietf-ip:ipv6": {}}, {"name": "Loopback9", "description": "999", "type": "iana-if-type:softwareLoopback", "enabled": true, "ietf-ip:ipv4": {"address": [{"ip": "10.9.9.9", "netmask": "255.255.255.0"}]}, {"ip": "172.29.0.9", "netmask": "255.255.255.0"}]}, "ietf-ip:ipv6": {}}]}}
```

RESTCONF Pretty Response with json.dumps

```
{
  "ietf-interfaces:interfaces": {
    "interface": [{
      "name": "GigabitEthernet1",
      "description": "VBox",
      "type": "iana-if-type:ethernetCsmacd",
      "enabled": true,
      "ietf-ip:ipv4": {
        "address": [{
          "ip": "192.168.56.101",
          "netmask": "255.255.255.0"
        }]
      },
      "ietf-ip:ipv6": {}
    }]
  }
}
```

2020 Cisco and/or its affiliates. All rights reserved.

RESTCONF Filter Response Data

PYTHON CODE TO FILTER JSON DATA FROM CSR1000v (no loop, only indexes used)

```
interface_name = resp["ietf-interfaces:interfaces"]["interface"][0]["name"]
ip = resp["ietf-interfaces:interfaces"]["interface"][0]["ietf-ip:ipv4"]["address"]

ip1 = resp["ietf-interfaces:interfaces"]["interface"][0]["ietf-ip:ipv4"]["address"][0]["ip"]
ip2 = resp["ietf-interfaces:interfaces"]["interface"][1]["ietf-ip:ipv4"]["address"][0]["ip"]
ip3 = resp["ietf-interfaces:interfaces"]["interface"][1]["ietf-ip:ipv4"]["address"][1]["ip"]
```

OUTPUT

=> Filtered response (first interface)

```
Interface Name          => GigabitEthernet1
IP Address + Mask       => [{'ip': '192.168.56.101', 'netmask': '255.255.255.0'}]
```

=> Filtered response (three interfaces)

```
192.168.56.101
10.9.9.9
172.29.0.9
```

Cisco DNA Center API Token

PYTHON API REQUEST

```
import requests
import json

### View the          to examine the API call URL and authentication parameters
url = "https://sandboxdnac.cisco.com/dna/system/api/v1/auth/token"
resp = requests.post( url, auth=auth, verify=False )
```

RAW RESPONSE

```
{ "Token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdWIiOiI1ZTlkYmI3NzdjZDQ3ZTAwNGM2N2RkMGUiLCJhdXRoU291cmNlIjoiaW50ZXJuYWwiLCJ0ZW5hbnROYW11IjoieVE5UMCI6Ixt1u9Vr9I_pj8EmkC3zIUSx5Hjr__TA-8VG86IGwW5-eTRRYaAcf2g8t6UkMs8Y9aGbfcDRgWfxmJOtPxx4_20J7tQIIgzQ9Iod9xY4UYCg8g6qu1DQuEoikWFLW_lH6aA" }
```

PYTHON FILTER to isolate token

```
token = resp["Token"]
print(token)
```

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdWIiOiI1ZTlkYmI3NzdjZDQ3ZTAwNGM2N2RkMGUiLCJhdXRoU291cmNlIjoiaW50ZXJuYWwiLCJ0ZW5hbnROYW11IjoieVE5UMCI6Ixt1u9Vr9I_pj8EmkC3zIUSx5Hjr__TA-8VG86IGwW5-eTRRYaAcf2g8t6UkMs8Y9aGbfcDRgWfxmJOtPxx4_20J7tQIIgzQ9Iod9xY4UYCg8g6qu1DQuEoikWFLW_lH6aA
```


DNA Center Response Data - Network Device List

Url: "https://sandboxdnac.cisco.com:443/dna/intent/api/v1/network-device"

Authentication header needed

JSON Response (partial)

```
{
  "response": [{
    "family": "Switches and Hubs",
    "hostname": "cat_9k_1",
    "macAddress": "f8:7b:20:67:62:80",
    "serialNumber": "FCW2136L0AK",
    "upTime": "25 days, 19:42:18.12",
    "softwareType": "IOS-XE",
    "softwareVersion": "17.3.1",
    "bootDateTime": "2020-12-24 17:11:54",
    "managementIpAddress": "10.10.22.66",
    "platformId": "C9300-24UX",
    "reachabilityStatus": "Reachable",
    "series": "Cisco Catalyst 9300 Series Switches",
    "type": "Cisco Catalyst 9300 Switch",
    "role": "ACCESS",
    "instanceUuid": "21335daf-f5a1-4e97-970f-ce4eaec339f6",
    "id": "21335daf-f5a1-4e97-970f-ce4eaec339f6"
  ]],
  "version": "1.0"
}
```

DNA Center Response Data - Network Device List

```
dev_list = [] #creating empty list

# looping through results and filter needed information
# creating new JSON structure

for device in resp_devices_json['response']:

    dev_dict = {} #create empty dict
    dev_dict['hostname'] = device['hostname']
    dev_dict['type'] = device['type']
    dev_dict['macAddress'] = device['macAddress']
    dev_dict['managementIpAddress'] = device['managementIpAddress']
    dev_dict['serialNumber'] = device['serialNumber']
    dev_dict['softwareType'] = device['softwareType']
    dev_dict['softwareVersion'] = device['softwareVersion']
    dev_dict['reachabilityStatus'] = device['reachabilityStatus']
    dev_list.append(dev_dict )
```

DNA Center Response Data - Client Health

DNA CENTER REQUEST

```
import requests
import json

# f-string is used for string formatting of request url
get_resp = requests.get(f"https://{host}/dna/intent/api/v1/client-health",
                        headers=headers, params=params)
```

RAW RESPONSE DATA FRAGMENT

```
{'response': [{'siteId': 'global', 'scoreDetail': [{'scoreCategory': {'scoreCategory': 'CLIENT_TYPE', 'value': 'ALL'}, 'scoreValue': 29, 'clientCount': 82, 'clientUniqueCount': 82, 'starttime': 1611071700000, 'endtime': 1611072000000}, {'scoreCategory': {'scoreCategory': 'CLIENT_TYPE', 'value': 'WIRED'}, 'scoreValue': 100, 'clientCount': 2, 'clientUniqueCount': 2, 'starttime': 1611071700000, 'endtime': 1611072000000, 'scoreList': [{'scoreCategory': {'scoreCategory': 'SCORE_TYPE', 'value': 'POOR'}, 'scoreValue': -1, 'clientCount': 0, 'clientUniqueCount': 0, 'starttime': 1611071700000, 'endtime': 1611072000000}, ...
```

DNA Center Filtering Response Data - Client Health

FILTERING DATA IN THE RESPONSE

```
print(type(get_resp_json))
```

```
print(get_resp_json["response"][0]["scoreDetail"][0]["clientCount"])
```

OUTPUT

-

```
Output 1: <class 'dict'>
```

-

```
Output 2:
```

```
Number of clients in first group: 82
```

/people/me

Webex API Call: "https://api.ciscospark.com/v1/people/me"

Partial Output (with some response data highlighted)

```
{
  "id": "Y21zY29zcGFya...N1YzQ5NjI5MGY",
  "emails": [
    {
      "value": "Yvan.rooseleer@biasc.be",
      "type": "personal-room",
      "value": "838744612@biasc.webex.com",
      "primary": false
    },
    {
      "value": "yvan.rooseleer@biasc.webex.com",
      "type": "personal-room",
      "value": "yvan.rooseleer@biasc.webex.com",
      "primary": false
    },
    {
      "value": "Yvan.rooseleer@biasc.calls.webex.com",
      "type": "cloud-calling",
      "value": "Yvan.rooseleer@biasc.calls.webex.com",
      "primary": true
    }
  ],
  "displayName": "Yvan Rooseleer",
  "nickName": "Yvan",
  "firstName": "Yvan",
  "lastName": "Rooseleer",
  "avatar": "https://avatar-prod-us-east-2.webexcontent.com/Avtr~V1~e4d4112d-2548-4a47-810e-04fe64-a79b-49c5-823a-92cec494bb9c1d1de6~1600",
  "orgId": "Y21zY29zcGFyazovL3VzL09SR0FOSctODEwZS0wNGZlNDVlYTE4MWY",
  "roles": [
    "Y21zY29zcGFya9hZG1pbG",
    "licenses": [
      "Y21tQtNDExMS1hYThkLTA1MDI3N2Y3Zjd1OQ",
      "Y21zY29zcGFyazovL3VzL0xjQ0VOy53ZWJleC5jb20",
      "Y21zY29zcGFyazovL3VzL0xjQ0VOUxYmJlNTU2LWQwZmItNGFiNy1hMTYyLTlmNjQ2OGIyYmU5ZA",
      "Y21zYVOU0UvZTRkNDExMmQtMjU0OC00MWBiNDI4MmQ5NmY5NA",
      "Y21zYTJhNTNfYmlhc2Mud2ViZXguY29t",
      "Y21zYZWExODFmOkZTU18xYjcyOGZmOS03ZGU4LTRjYjctOTU0MC0yOTMyMGI1YTQyY2I",
      "Y21zY29zUtMDRmZTQ1ZWExODFmOkZUTV9mNWZkZTM1Zi00NzA0LTQ2MGEtODEwZi00YzVzMzUyNDFlNjk",
      "Y21zY29zcExODFmOkNHXzVkJyYjYyLWNmYTItNGFjZC04MTRlLTgwYjNinNWVkjNlZA",
      "Y21zY29zcGFyazovL3jQ3MzgyOC1hOTgwLTQ3MmYtODE5ZC02Yj1jY2UwOGU5MmI",
      "Y21zY29zcGFyazovLmOkZNU185ZWZhNzgxNC0zMzEzLTQ2NGYtOTY0Mi0wMjM5ODc1YmM5Zjg",
      "Y21zY29zcGFyazovLNDExMmQtMjU0OGE3NS1hNDNmLTBkYmJhMjIyNzg3Zl9iaWFzYy53ZWJleC5jb20",
      "Y21mMwMjFkLTgwYjctNDFiYi1iZThhLWM0YjFiZjcyNTE4YV9iaWFzYy53ZWJleC5jb20",
      "Y21zY29zcGFyazovL3V0YTQ3LTGxMGUtMDRmZTQ1ZWExODFmOk1kYmFiMDcxYmY0NA",
      "Y21zY29zcGFyazovL3VzL0xjQ0VOUxZnNkMDU3N2RiLTFjOGItNDQ4My1hMTBjLzYy53ZWJleC5jb20",
      "created": "2016-12-23T08:38:22.877Z",
      "lastModified": "2021-01-26T17:55:07.662Z",
      "lastActivity": "2021-01-26T17:54:24.481Z",
      "status": "active",
      "invitePending": false,
      "loginEnabled": true,
      "type": "person",
      "trainSiteNames": ["biasc.webex.com"]
    }
  ]
}
```

Webex Filtering Response Data

Display Filtered Results From Webex

```
print("Name: " + resp['displayName'])  
print("Created: " + resp['created'])  
print("User Type: " + resp['type'])  
print("User Status: " + resp['status'])
```

Output

Displaying partial information

Name: Yvan Rooseleer

Created: 2016-12-23T08:38:22.877Z

User Type: person

User Status: active

Ansible Response Data

Ansible command to gather facts from the webserver inventory

```
$ ansible webserver -m gather_facts --tree ./tmp_facts
```

URL for the complete response file:

Partial Output (with some response data highlighted)

```
{
  "ansible_facts": {
    "ansible_all_ipv4_addresses": [
      "192.0.2.1",
      "192.0.2.2",
      "192.0.2.3",
      "192.0.2.4",
      "192.0.2.5",
      "10.0.2.15",
      "172.17.0.1"
    ],
    "ansible_all_ipv6_addresses": [
      "fe80::9002:c8ff:fee8:bb09",
      "fe80::3c67:a5ff:fe17:e4cf",
      "fe80::a00:27ff:fee9:3de6",
      "fe80::42:3ff:fef6:9477"
    ],
    "ansible_apparmor": {
      "status": "enabled"
    },
    "ansible_architecture": "x86_64",
    "ansible_bios_date": "12/01/2006",
    "ansible_bios_version": "VirtualBox",
    "ansible_cmdline": {
      "BOOT_IMAGE": "/boot/vmlinuz-5.4.0-37-generic",
      "quiet": true,
      "ro": true,
      "root": "UUID=fb261367-cf98-4bce-b682-42b3de0a8ab9",
      "vga": "792",
      "zswap.enabled": "1"
    },
    "ansible_date_time": {
      "date": "2021-01-20",
      "day": "20",
      "epoch": "1611160850",
      "hour": "16",
      "iso8601": "2021-01-20T16:40:50Z",
      "iso8601_basic": "20210120T164050181658",
      "iso8601_basic_short": "20210120T164050",
      "iso8601_micro": "2021-01-20T16:40:50.181774Z",
      "minute": "40",
      "month": "01",
      "second": "50",
      "time": "16:40:50",
      "tz": "UTC",
      "tz_offset": "+0000",
      "weekday": "Wednesday",
      "weekday_number": "3",
      "weeknumber": "03",
      "year": "2021"
    },
    "ansible_default_ipv4": {
      "address": "10.0.2.15",
      "alias": "enp0s3",
      "broadcast": "10.0.2.255",
      "gateway": "10.0.2.2",
      "interface": "enp0s3",
      "macaddress": "08:00:27:e9:3d:e6",
      "mtu": 1500,
      "netmask": "255.255.255.0",
      "network": "10.0.2.0",
      "type": "ether"
    },
    "ansible_distribution": "Ubuntu",
    "ansible_distribution_file_parsed": true,
    "ansible_distribution_file_path": "/etc/os-release",
    "ansible_distribution_file_variety": "Debian",
    "ansible_distribution_major_version": "20",
    "ansible_distribution_release": "focal",
    "ansible_distribution_version": "20.04",
    "ansible_dns": {
      "nameservers": [
        "127.0.0.53"
      ]
    },
    "ansible_nodename": "labvm",
    "ansible_os_family": "Debian",
    "ansible_pkg_mgr": "apt",
    "ansible_proc_cmdline": {
      "BOOT_IMAGE": "/boot/vmlinuz-5.4.0-37-generic",
      "quiet": true,
      "ro": true,
      "root": "UUID=fb261367-cf98-4bce-b682-42b3de0a8ab9",
      "vga": "792",
      "zswap.enabled": "1"
    },
    "ansible_processor": [
      "0",
      "GenuineIntel",
      "Intel(R) Core(TM) i7-7600U CPU @ 2.80GHz"
    ],
    "ansible_processor_cores": 2,
    "ansible_processor_count": 1,
    "ansible_processor_threads_per_core": 1,
    "ansible_processor_vcpus": 2,
    "ansible_product_name": "VirtualBox",
    "ansible_product_serial": "NA",
    "ansible_product_uid": "NA",
    "ansible_product_version": "1.2",
    "ansible_python": {
      "executable": "/usr/bin/python3",
      "has_sslcontext": true,
      "type": "cpython",
      "version": {
        "major": 3,
        "micro": 2,
        "minor": 8,
        "releaselevel": "final",
        "serial": 0
      },
      "version_info": [
        3,
        8,
        2,
        "final",
        0
      ]
    },
    "ansible_python_version": "3.8.2",
    "ansible_real_group_id": 900,
    "ansible_real_user_id": 900,
    "ansible_selinux": {
      "status": "disabled"
    },
    "ansible_selinux_python_present": true,
    "ansible_service_mgr": "systemd",
    "ansible_virtualization_role": "guest",
    "ansible_virtualization_type": "virtualbox",
    "discovered_interpreter_python": "/usr/bin/python3",
    "gather_subset": [
      "all"
    ],
    "module_setup": true,
    "changed": false,
    "deprecations": [],
    "warnings": []
  }
}
```

Ansible Response Data

Parsing And Filtering Ansible JSON Data -- Keys Level 1

Code

```
ansible_dict = json.loads(ansible_json_doc)
print(ansible_dict.keys())
```

Output (Level 1)

```
['ansible_facts', 'changed', 'deprecations', 'warnings']
```


Ansible Response Data

Parsing And Filtering Ansible JSON Data -- Keys Level 2

Code

```
print(ansible_dict['ansible_facts'].keys())
```

Output (Level 2)

```
['ansible_all_ipv4_addresses', 'ansible_all_ipv6_addresses',  
'ansible_default_ipv4', 'ansible_default_ipv6', 'ansible_distribution',  
'ansible_distribution_file_variety', 'ansible_distribution_major_version',  
'ansible_distribution_release', 'ansible_distribution_version', 'ansible_env',  
'ansible_kernel', 'ansible_kernel_version', 'ansible_python_version']
```

Ansible Response Data

Parsing And Filtering Ansible JSON Data -- Keys Level 2

Code

```
O1 = ansible_dict["ansible_facts"]["ansible_distribution"]  
O2 = ansible_dict["ansible_facts"]["ansible_distribution_release"]  
O3 = ansible_dict["ansible_facts"]["ansible_distribution_version"]
```

Output Level 2

```
Ansible Distribution: Ubuntu  
Ansible Distribution Release: focal  
Ansible Distribution Version: 20.04
```

Ansible Response Data

Parsing And Filtering Ansible JSON Data -- Keys Level 2 and 3

Code

```
print("Ansible Kernel: " + ansible_dict["ansible_facts"]["ansible_kernel"])
print("Ansible Home: " + \
ansible_dict["ansible_facts"]["ansible_env"]["HOME"])
print("Ansible User: " + \
ansible_dict["ansible_facts"]["ansible_env"]["USER"])
print("IP Address: " + \
ansible_dict["ansible_facts"]["ansible_default_ipv4"]["address"])
```

Output Level 2-3

Output Ansible Kernel: 5.4.0-37-generic

Output Ansible Home: /home/devasc

Output Ansible User: devasc

Output IP Address: 10.0.2.15

Docker Response Data -- docker image (partial)

\$ docker image inspect ubuntu

Output (with some response data highlighted)

```
[{"Id": "sha256:9140108b62dc87d9b278bb0d4fd6a3e44c2959646eb966b86531306faa81b09b", "RepoTags": ["ubuntu:latest"], "RepoDigests":  
["ubuntu@sha256:bc2f7250f69267c9c6b66d7b6a81a54d3878bb85f1ebb5f951c896d13e6ba537"], "Parent": "", "Comment": "", "Created": "2020-09-  
25T22:34:30.295807036Z", "Container": "1046a5d685aef5c37d1829040ca8083b94e4c069ca4963f4b16a6ade2e077b06", "ContainerConfig": {"Hostname":  
"1046a5d685ae", "Domainname": "", "User": "", "AttachStdin": false, "AttachStdout": false, "AttachStderr": false, "Tty": false,  
"OpenStdin": false, "StdinOnce": false, "Env": ["PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"], "Cmd": ["/bin/sh",  
"-c", "#(nop) ", "CMD ["/bin/bash\""], "ArgsEscaped": true, "Image":  
"sha256:4ff2090064e7e38688bce713d50f3202d227b3c89feceal434271c912ccd47e0", "Volumes": null, "WorkingDir": "", "Entrypoint": null,  
"OnBuild": null, "Labels": {}, "DockerVersion": "18.09.7", "Author": "", "Config": {"Hostname": "", "Domainname": "", "User": "",  
"AttachStdin": false, "AttachStdout": false, "AttachStderr": false, "Tty": false, "OpenStdin": false, "StdinOnce": false, "Env":  
["PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"], "Cmd": ["/bin/bash"], "ArgsEscaped": true, "Image":  
"sha256:4ff2090064e7e38688bce713d50f3202d227b3c89feceal434271c912ccd47e0", "Volumes": null, "WorkingDir": "", "Entrypoint": null,  
"OnBuild": null, "Labels": null}, "Architecture": "amd64", "Os": "linux", "Size": 72875723, "VirtualSize": 72875723, "GraphDriver":  
{"Data": {"LowerDir": "/var/lib/docker/overlay2/ 5d66f43ef0d92430a195045c4715ff3d49ee88ccb0bb7f6e69ccc5d60fde0ece  
/diff:/var/lib/docker/overlay2/0b75d53292ccb20230879e49c7fe6eeal57adbd083d414ba3d0485959cd35086/diff", "MergedDir":  
"/var/lib/docker/overlay2/c3bab8487f590bfd66e53db0b1902ab5a8b8fbfa1f02cfc3b4f0f9fa25158f2d/merged", "UpperDir":  
"/var/lib/docker/overlay2/c3bab8487f590bfd66e53db0b1902ab5a8b8fbfa1f02cfc3b4f0f9fa25158f2d/diff", "WorkDir":  
"/var/lib/docker/overlay2/c3bab8487f590bfd66e53db0b1902ab5a8b8fbfa1f02cfc3b4f0f9fa25158f2d/work"}, "Name": "overlay2"}, "RootFS":  
{"Type": "layers", "Layers": ["sha256:d42a4fdf4b2ae8662ff2ca1b695eae571c652a62973c1beb81a296a4f4263d92",  
"sha256:90ac32a0d9ab11e7745283f3051e990054616d631812ac63e324c1a36d2677f5",  
"sha256:782f5f011ddaf2a0bfd38cc2ccabd634095d6e35c8034302d788423f486bb177"]}, "Metadata": {"LastTagTime": "0001-01-01T00:00:00Z"}}]
```

Docker Response Data

Code Example

```
import json

### Converting json string to dict
docker_dict = json.loads(docker_json_file)

### Filtering Response Data from dict
print(docker_dict[0]["Created"])
print(docker_dict[0]["Architecture"])
print(docker_dict[0]["Os"])
```

Output

```
Created      => 2020-09-25T22:34:30.295807036Z
Architecture => Amd64
OS           => Linux
```

Docker Response Data -- docker image (partial)

```
$ docker network inspect \
bridge
```

```
[
  {
    "Name": "bridge",
    "Id": "566a72fc961157e2e71cc257fc2132beebc491a712967ef42bddcab70cbdbb23",
    "Created": "2020-12-09T17:51:15.816558163Z",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
        {
          "Subnet": "172.17.0.0/16",
          "Gateway": "172.17.0.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
      "4e99a64e10dfcf6608ald47f4349676c745bf234cebd52826d786db9a3be2811": {
        "Name": "samplerunning",
        "EndpointID": "22kbd3fa7e76635c3172446813fe5104537c8f69c6c23474272b379dede44fe7",
        "MacAddress": "02:42:ac:11:00:03",
        "IPv4Address": "172.17.0.3/16",
        "IPv6Address": ""
      }
    },
    "Options": {
      "com.docker.network.bridge.default_bridge": "true",
      "com.docker.network.bridge.enable_icc": "true",
      "com.docker.network.bridge.enable_ip_masquerade": "true",
      "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
      "com.docker.network.bridge.name": "docker0",
      "com.docker.network.driver.mtu": "1500"
    },
    "Labels": {}
  }
]
```

Python Script - Filtering Response Data from Docker

Code Example

```
import json

### Converting json string to dict, and showing keys at level 1
docker_net_dict = json.loads(docker_json_file2)

### Filtering from dict
print(docker_net_dict[0]["Name"])
print(docker_net_dict[0]["Created"])
print(docker_net_dict[0]["Containers"]["4e9a64e...bd52b811"] ["IPv4Address"])
```

Output

```
bridge
2020-12-09T17:51:15.816558163Z
172.17.0.3/16
```

Exercise - Webex Groups Dict Example

```
groups_struct = {
  "groups": [
    { "group": { "group_id": "G-A"
      , "group_name": "DEVASC_A" ,
      "members": [
        {"person_id": "P-1" , "person_name": "Noel", "email": "noel@odisee.be"},
        {"person_id": "P-2" , "person_name": "Mary", "email": "mary@odisee.be"},
        {"person_id": "P-3" , "person_name": "Jens", "email": "jens@odisee.be"}
      ]
    }
  },
  { "group": { "group_id": "G-B"
    , "group_name": "DEVASC_B" ,
    "members": [
      {"person_id": "P-4" , "person_name": "Ives", "email": "ives@odisee.be"},
      {"person_id": "P-5" , "person_name": "John", "email": "john@odisee.be"},
      {"person_id": "P-6" , "person_name": "Alec", "email": "alec@odisee.be"}
    ]
  }
}
}
```

© 2020 Cisco and/or its affiliates. All rights reserved.

Exercise - Webex Groups Python Script

Code Example ### First group with first member, second group with first member

```
### Select first group, first person
```

```
resp_a1 = groups_struct["groups"][0]["group"]["group_name"]
```

```
resp_a2 = groups_struct["groups"][0]["group"]["members"][0]["person_name"]
```

```
### Select second group, only the first person
```

```
resp_b1 = groups_struct["groups"][1]["group"]["group_name"]
```

```
resp_b2 = groups_struct["groups"][1]["group"]["members"][0]["person_name"]
```

Output

First group, First person

DEVASC_A => Noel

Second group, First person

DEVASC_B => Ives

And so on ... (more groups, more members)

© 2020 Cisco and/or its affiliates. All rights reserved.

Exercise - Webex Groups Python Script

Code Example ### All groups with members and email addresses

```
for g in groups_struct["groups"]:  
    print(g["group"]["group_name"])  
    for p in g["group"]["members"]:  
        print(p["person_name"] + " => " + p["email"])
```

Output

DEVASC_A

```
Noel => noel@odisee.be  
Mary => mary@odisee.be  
Jens => jens@odisee.be
```

DEVASC_B

```
Ives => ives@odisee.be  
John => john@odisee.be  
Alec => alec@odisee.be
```

© 2020 Cisco and/or its affiliates. All rights reserved.

And so on ... (more groups)

Loop Exercise 2 - Network Devices Dict Example

```
rack_struc = {  
    "rack": [  
        { "device": { "dev_id": "D1" ,  
                      "dev_name": "R1" ,  
                      "role": "router" ,  
                      "interfaces": [  
                          { "interface": "GigabitEthernet1" ,  
                            "ipaddress": "10.0.1.1",  
                            "subnet_mask": "255.255.255.0"},  
                          { "interface": "GigabitEthernet2" ,  
                            "ipaddress": "10.0.3.1",  
                            "subnet_mask": "255.255.255.0"},  
                          { "interface": "GigabitEthernet3" ,  
                            "ipaddress": "10.0.4.1",  
                            "subnet_mask": "255.255.255.0"}  
                      ]  
        }  
    ]  
}
```

} © 2020 Cisco and/or its affiliates. All rights reserved.

And so on ... (if more network devices)

Loop Exercise 2 - Network Devices Python Script

Code Sample ### All network devices interfaces and ip addresses

```
for g in rack_struct["rack"]:  
    print(g["device"]["dev_name"])  
    for p in g["device"]["interfaces"]:  
        print(p["interface"]+" => "+p["ipaddress"])
```

Output

R1

GigabitEthernet1 => 10.0.1.1

GigabitEthernet2 => 10.0.3.1

GigabitEthernet3 => 10.0.4.1

C1

VLAN1 => 10.0.1.2

VLAN2 => 10.0.2.1

VLAN20 => 10.0.20.1

AC

VLAN2 => 10.0.2.2

© 2020 Cisco and/or its affiliates. All rights reserved.

And so on ... (if more network devices)

Python & JSON Workshop Part 2

Part 1: DevNet Associate: Filtering JSON Data

Description: This part covers some aspects of the DevNet Associate course. Join to learn how to interpret JSON data coming from Webex Teams API, DNAC API, Docker inspect, Ansible.

Part 2: DevNet Associate: Generating JSON Data

Description: This part covers some aspects of the DevNet Associate course. Join to learn how to generate JSON data for Webex Teams API, IP Devices and Network Services from a source in the Excel format.

Problem Statement

Create Webex spaces and members from Source Data in Excel Format

Transform 2-dimensional structure into hierarchical tree structure

Simpler than YANG Model: less hierarchical levels

	A	B	C
1	group	name	email
2	GROUP_ALPHA	Vincent Cassata	vincent.cassata@student.bxl.be
3	GROUP_ALPHA	Giovanni Di Tulio	Giovanni.ditullio@student.bxl.be
4	GROUP_ALPHA	Milan Vandeveld	milan.vandeveld@student.bxl.be
5	GROUP_ALPHA	Tomas Vertessen	tomas.vertessen@student.bxl.be
6	GROUP_ALPHA	Mehdi Dahli	mehdi.dahli@student.bxl.be
7	GROUP_KAPPA	Ur Salangpour	ur.salangpour@student.bxl.be
8	GROUP_KAPPA	Mon Gallin	mon.gallin@student.bxl.be
9	GROUP_KAPPA	Artur Ikiya	artur.ikiya@student.bxl.be
10	GROUP_KAPPA	Bram Vanbever	bram.vanbever@student.bxl.be
11	GROUP_KAPPA	JR Ibara	jr.ibara@student.bxl.be
12	GROUP_DELTA	Jona Ferbiest	jona.ferbiest@student.bxl.be
13	GROUP_DELTA	Bart Siperius	bart.siperius@student.bxl.be
14	GROUP_DELTA	Joren Huysegoms	joren.huysegoms2@student.bxl.be
15	GROUP_DELTA	Sam Bulduk	sam.bulduk@student.bxl.be
16	GROUP_DELTA	Ferre Van Malder	ferre.vanmalder@student.bxl.be
17	GROUP_DELTA	Mikail Defossez	mikail.defossez@student.bxl.be
18	* names are non-existent		

```
{
  "groups": [{
    "group": {
      "group_name": "GROUP_ALPHA",
      "members": [{
        "person_name": "Vincent Cassata",
        "email": "vincent.cassata@student.bxl.be"
      },
      {
        "person_name": "Giovanni Di Tulio",
        "email": "Giovanni.ditullio@student.bxl.be"
      },
      {
        "person_name": "Milan Vandeveld",
        "email": "milan.vandeveld@student.bxl.be"
      }, ...
    ]
  }
}
```

Task 1 - Source Spreadsheet Webex Groups

	A	B	C
1	group	name	email
2	GROUP_ALPHA	Vincent Cassata	vincent.cassata@student.bxl.be
3	GROUP_ALPHA	Giovanni Di Tulio	Giovanni.ditullio@student.bxl.be
4	GROUP_ALPHA	Milan Vandeveld	milan.vandeveld@student.bxl.be
5	GROUP_ALPHA	Tomas Vertessen	tomas.vertessen@student.bxl.be
6	GROUP_ALPHA	Mehdi Dahli	mehdi.dahli@student.bxl.be
7	GROUP_KAPPA	Ur Salangpour	ur.salangpour@student.bxl.be
8	GROUP_KAPPA	Mon Gallin	mon.gallin@student.bxl.be
9	GROUP_KAPPA	Artur Ikiya	artur.ikiya@student.bxl.be
10	GROUP_KAPPA	Bram Vanbever	bram.vanbever@student.bxl.be
11	GROUP_KAPPA	JR Ibara	jr.ibara@student.bxl.be
12	GROUP_DELTA	Jona Ferbiest	jona.ferbiest@student.bxl.be
13	GROUP_DELTA	Bart Siperius	bart.siperius@student.bxl.be
14	GROUP_DELTA	Joren Huysegoms	joren.huysegoms2@student.bxl.be
15	GROUP_DELTA	Sam Bulduk	sam.bulduk@student.bxl.be
16	GROUP_DELTA	Ferre Van Malder	ferre.vanmalder@student.bxl.be
17	GROUP_DELTA	Mikail Defossez	mikail.defossez@student.bxl.be
18	* names are non-existent		

Business Context

At a recent meeting it was decided that a number of new **Webex** groups should be created, each with several members. A member has a name and an email address.

The **spreadsheet** on the left was created by communication and sent to Webex admin.

The decision was made that **JSON** should be used as an intermediate format to automate the creation of new groups and memberships.

Task 1 - Target Structure Webex Groups

```
groups_struct = {
  "groups": [
    { "group": { "group_id": "G-A"
                  , "group_name": "DEVASC_A" ,
                  "members": [
                    { "person_id": "P-1" , "person_name": "Noel", "email": "noel@odisee.be"},
                    { "person_id": "P-2" , "person_name": "Mary", "email": "mary@odisee.be"},
                    { "person_id": "P-3" , "person_name": "Jens", "email": "jens@odisee.be"}
                  ]
                }
      },
    { "group": { "group_id": "G-B"
                  , "group_name": "DEVASC_B" ,
                  "members": [
                    { "person_id": "P-4" , "person_name": "Ives", "email": "ives@odisee.be"},
                    { "person_id": "P-5" , "person_name": "John", "email": "john@odisee.be"},
                    { "person_id": "P-6" , "person_name": "Alec", "email": "alec@odisee.be"}
                  ]
                }
      }
  ]
}
```

} © 2020 Cisco and/or its affiliates. All rights reserved.

Task 1 - Step 1: Define Python Data Rules

REWRITING RULES TO GENERATE THE DATA STRUCTURE

Most of the time you will have to manage structures of type **dict** and **list**. These are very common for **JSON data exchange**.

```
member_dict    =>  {"person_name": "x", "email": "y", "group": "z"}
member_list    =>  [member_dict]

group_dict     =>  {group_name, member_list} | {group_name, [member_dict]}
group_list     =>  [group_dict]

groups_struct  =>  {group_list} | {[group_dict]}
```

Task 1 - Step 2: Read Two Excel Records

Simplified code: converting Excel into Python dict

```
import xlrd    # library to manage excel spreadsheets
import json    # library to manage JSON classes and functions

wb = xlrd.open_workbook("webex_groups.xlsx")
sheet = wb.sheet_by_index(0)  # read data from the first tab

member_dict["group"]          = sheet.cell_value(1, 0)
member_dict["person_name"]    = sheet.cell_value(1, 1)
member_dict["email"]          = sheet.cell_value(1, 2)

member_dict["group"]          = sheet.cell_value(2, 0)
member_dict["person_name"]    = sheet.cell_value(2, 1)
member_dict["email"]          = sheet.cell_value(2, 2)
```

Result Example in Python dict format:

```
{'group': 'GROUP_ALPHA', 'person_name': 'Vincent Cassata', 'email': 'vincent.cassata@student.bxl.be'}
{'group': 'GROUP_ALPHA', 'person_name': 'Giovanni Di Tulio', 'email': ' '}
```

Task 1 - Step 3: Read All Excel Records (loop)

Simplified code: converting Excel data into Python dict

```
import xlrd    # library to manage excel spreadsheets
import json    # library to manage JSON classes and functions
```

```
def find_all_persons_and_groups(xlf):
    wb = xlrd.open_workbook(xlf)
    sheet = wb.sheet_by_index(0)
    number_rows = sheet.nrows
    member_list = []
    for r in range(number_rows):
        if r > 0: ### first row contains column names
            member_dict["group"] = sheet.cell_value(r, 0)
            member_dict["person_name"] = sheet.cell_value(r, 1)
            member_dict["email"] = sheet.cell_value(r, 2)
            member_list.append(member_dict.copy())
    return member_list
```

Function Call in Python Script

```
member_list = find_all_persons_and_groups("webex_groups.xlsx")
```

Result Example:

```
[{'group': 'GROUP_ALPHA', 'person_name': 'Vincent Cassata', 'email':
'vincent.cassata@student.bxl.be'}, {'group': 'GROUP_ALPHA', 'person_name': 'Giovanni Di Tulio',
'email': 'Giovanni .ditullio@student.bxl.be'}, ... ]
```

Task 1 - Step 4: Create Structure Level 1

Simplified code: making list of groups from Python dict

```
def make_list_of_groups(member_list):  
    group_list = []  
    for rec in member_list:  
        group_list.append(rec["group"])  
    return group_list # => return the list of groups
```

Function Call in Python Script

```
group_list = make_list_of_groups(member_list)
```

Result Example: list of groups

```
['GROUP_ALPHA', 'GROUP_KAPPA', 'GROUP_DELTA']
```

Task 1 - Step 5: Create Structure Level 2

Simplified code: attaching group members to group

```
def attach_members_to_groups(group_rec, member_list):  
    mem_dict = {}  
    mem_list = [mem_dict]  
    for membr in member_list:  
        if membr["group"] == group_rec:  
            mem_dict["person_name"] = membr["person_name"]  
            mem_dict["email"] = membr["email"]  
            mem_list.append(mem_dict.copy())  
    return mem_list
```

Function Call in Python Script:

```
for group_rec in group_list:  
    all_members = attach_members_to_groups(group_rec, member_list)
```

Result Example

```
{'group': {'group': {'group_name': 'GROUP_ALPHA', 'members': [{'person_name': 'Vincent Cassata', 'email':  
'vincent.cassata@student.bxl.be'}, {'person_name': 'Giovanni Di Tulio', 'email':  
'Giovanni.ditullio@student.bxl.be'}, ...
```

Task 1 - Step 6: Run main() function

Simplified code: calling functions creating data structure

```
def main():  
    member_list = find_all_persons_and_groups("webex_groups.xlsx")  
    group_list = make_list_of_groups(member_list)  
    all_members = []  
    for group_rec in group_list:  
        all_members = attach_members_to_groups(group_rec, member_list)  
        group_dict["group"] = {"group": {"group_name": group_rec, "members": all_members }}  
        groups_struc["groups"].append(group_dict["group"])  
    js_groups = json.dumps(groups_struc)
```

Function Call in Python Script

execute main() when called directly

```
if __name__ == '__main__':  
    main()
```

Result Example

see next slide

Task 1 - Step 7: Verify or Validate Results (RAW)

```
{"groups": [{"group": {"group_name": "GROUP_ALPHA", "members": [{"person_name": "Vincent  
Cassata", "email": "vincent.cassata@student.bxl.be"}, {"person_name": "Giovanni Di Tulio",  
"email": "Giovanni.ditullio@student.bxl.be"}, {"person_name": "Milan Vandevelde", "email":  
"milan.vandevelde@student.bxl.be"}, {"person_name": "Tomas Vertessen", "email":  
"tomas.vertessen@student.bxl.be"}, {"person_name": "Mehdi Dahli", "email":  
"mehdi.dahli@student.bxl.be"}]}}, {"group": {"group_name": "GROUP_KAPPA", "members":  
[{"person_name": "Ur Salangpour", "email": "ur.salangpour@student.bxl.be"}, {"person_name":  
"Mon Gallin", "email": "mon.gallin@student.bxl.be"}, {"person_name": "Artur Ikiya",  
"email": "artur.lkiya@student.bxl.be"}, {"person_name": "Bram Vanbever", "email":  
"bram.vanbever@student.bxl.be"}, {"person_name": "JR Ibara", "email":  
"jr.ibara@student.bxl.be"}]}}, {"group": {"group_name": "GROUP_DELTA", "members":  
[{"person_name": "Jona Ferbiest", "email": "jona.ferbiest@student.bxl.be"}, {"person_name":  
"Bart Siperius", "email": "bart.siperius@student.bxl.be"}, {"person_name": "Joren  
Huysegoms", "email": "joren.huysegoms2@student.bxl.be"}, {"person_name": "Sam Bulduk",  
"email": "sam.bulduk@student.bxl.be"}, {"person_name": "Ferre Van Malder", "email":  
"ferre.vanmalder@student.bxl.be"}, {"person_name": "Mikail Defossez", "email":  
"mikail.defossez@student.bxl.be"}]}}]}
```

Task 1 - Step 7b: Verify or Validate Results (Tree)

```
{
  "groups": [{
    "group": {
      "group_name": "GROUP_ALPHA",
      "members": [{
        "person name": "Vincent Cassata",
        "email": "vincent.cassata@student.bxl.be"
      },
      {
        "person name": "Giovanni Di Tulio",
        "email": "Giovanni.ditullio@student.bxl.be"
      },
      {
        "person name": "Milan Vandevelde",
        "email": "milan.vandevelde@student.bxl.be"
      }, ...
    ]
  }
}
```

© 2020 Cisco and/or its affiliates. All rights reserved.

Task 2 - Source Spreadsheet Network Devices

	A	B	C	D	E
1	device	role	interface	ipaddress	subnetmask
2	RTR1	router	GigabitEthernet 0	192.0.2.254	255.255.255.0
3	RTR1	router	GigabitEthernet 1	10.0.1.1	255.255.255.0
4	RTR1	router	GigabitEthernet 2	10.0.2.1	255.255.255.0
5	MLS1	core switch	VLAN 1	10.0.1.2	255.255.255.0
6	MLS1	core switch	VLAN 2	10.0.2.1	255.255.255.0
7	MLS2	core switch	VLAN 1	10.0.1.3	255.255.255.0
8	MLS2	core switch	VLAN 2	10.0.2.2	255.255.255.0
9	ASW2	access switch	VLAN 1	10.0.1.2	255.255.255.0
10	ASW3	access switch	VLAN 1	10.0.1.3	255.255.255.0
11	ASW4	access switch	VLAN 1	10.0.1.4	255.255.255.0
12	ASW5	access switch	VLAN 1	10.0.1.5	255.255.255.0
13	ASW6	access switch	VLAN 2	10.0.2.6	255.255.255.0
14	ASW7	access switch	VLAN 2	10.0.2.7	255.255.255.0
15	ASW8	access switch	VLAN 2	10.0.2.8	255.255.255.0
16	ASW9	access switch	VLAN 2	10.0.2.9	255.255.255.0

Business Context

This **spreadsheet** was created by IT staff and sent to the Networking Team.

There are eleven **network devices** that need to be configured with one or more interfaces and ip addresses.

The task is to transform the spreadsheet into **JSON format** and to transmit the resulting data to a network automation tool.

The **automation tool** is able to accept and transform JSON data.

Task 2 - Target Structure Network Devices - 2 levels

```
{
  "rack": [{
    "device": {
      "dev_name": "RTR1",
      "role": "router"
    },
    "interfaces": [{
      "interface": "GigabitEthernet 0",
      "ipaddress": "192.0.2.254",
      "subnetmask": "255.255.255.0"
    },
    {
      "interface": "GigabitEthernet 1",
      "ipaddress": "10.0.1.1",
      "subnetmask": "255.255.255.0"
    },
    {
      "interface": "GigabitEthernet 2",
      "ipaddress": "10.0.2.1",
      "subnetmask": "255.255.255.0"
    }
  ]
}, ...
}
```

=> Develop a Python script based on the example explained in task 1.

Task 2 - Target Structure Network Devices - YANG

```
{  
  "ietf-interfaces:interfaces": {  
    "interface": [{  
      "name": "GigabitEthernet1",  
      "description": "VBox",  
      "type": "iana-if-type:ethernetCsmacd",  
      "enabled": true,  
      "ietf-ip:ipv4": {  
        "address": [{  
          "ip": "192.168.56.101",  
          "netmask": "255.255.255.0"  
        }]  
      },  
      "ietf-ip:ipv6": {}  
    },  
    {  
      "name": "Loopback9",  
      "description": "Lo9",  
      "type": "iana-if-type:softwareLoopback",  
      "enabled": true,  
      "ietf-ip:ipv4": {  
        "address": [{  
          "ip": "10.9.9.9",  
          "netmask": "255.255.255.0"  
        }]  
      },  
      "ietf-ip:ipv6": {}  
    }  
  ]  
}
```

Time for Questions & Remarks



1	2	3	4	5
Class Activities	Study & Prep	Practical Activities		
<u>Teaching & Evaluating</u>	<u>Open Learning Center</u>	<u>Team Work</u>		
Structuring	Self-Study	Virtual hands-on labs		
Planning	Self-Evaluation	Remote hands-on labs		
Briefings	Simulation exercises	Physical hands-on labs		
Formal Evaluation	Preparations	Case Studies & Projects		
Debriefings		Work Placement & Internships		
2/5 - 80 h	1/5 - 40 h	2/5 - 80 h		

References

Slide Deck for This Workshop

URL: https://docs.google.com/presentation/d/1_yR5CA_V2rDrvUag58gyCHxQncATbBoBxxLTf_4NK_M/edit?usp=sharing

Data Files for the Practical Examples and Exercises of This Workshop

URL: <https://drive.google.com/drive/folders/1lp6nw4uxTHb5t9TxHEC3xcRPGzsTgEnH?usp=sharing>

Background Document For This Workshop

URL: <https://docs.google.com/document/d/1jWNaW4OMkcCu1wnXZbQSMKxy23dlcq6ds4CBnEEXe0Q/edit?usp=sharing>

My Recent Badges

Awards



[DevNet Associate](#)
[Instructor](#)
[Pioneer](#)

Cisco



DevNet Associate

Cisco



Cisco Certified
DevNet Associate

Cisco



DevNet Class of 2020

Cisco



Understanding
of Cisco
Network
Devices

Cisco



Understanding
Cisco Network
Security

Cisco



Cisco Certified
CyberOps
Associate

Cisco



Instructor 20
Years of Service

Cisco



Cisco Certified
Network
Associate
Security (CCN...

Cisco