# Devnet Sandbox

This lab was written to be run using the [DevNet Devbox Sandbox](https://devnetsandbox.cisco.com/RM/Diagram/Index/f1a51f3b-3377-444d-97f0-5ad300d976be?diagramType=Topology). This sandbox is a basic CentOS 7 workstation with typical development tools and software installed. Specifically used in this lab are Python 3.6 and Vagrant (used to instantiate an IOS XE router for use in the labs.)

If you are doing this lab on your own, you’ll need to reserve an instance of this sandbox before beginning. If you are doing this as part of a guided workshop, the instructor will assign you a pod.

## Steps to complete preparation

1. Using either AnyConnect or OpenConnect, establish a VPN to your pod.
2. SSH to the Devbox at IP 10.10.20.20 using credentials root / cisco123

ssh root@10.10.20.20

1. Install the Python 3.6 development libraries.

yum install -y python36u-devel

1. Add the IOS XE 16.9 Vagrant Box to your workstation. Instructions for creating the Box file are available on github at [github.com/hpreston/vagrant\_net\_prog](https://github.com/hpreston/vagrant_net_prog/tree/master/box_building#cisco-csr-1000v). If you are completing this as part of a guided lab, the instructor will provide details on how to complete this step.

vagrant box add --name iosxe/16.09.01 serial-csr1000v-universalk9.16.09.01.box

1. Clone the code samples to the devbox from GitHub and change into the directory.

git clone https://github.com/hpreston/python\_networking  
cd python\_networking

1. Create a Python 3.6 virtual environment and install Python libraries for exercises.

python -m venv venv  
source venv/bin/activate  
pip install -r requirements.txt

1. Start and baseline the IOS XE Vagrant environment.

vagrant up

After it completes

python vagrant\_device\_setup.py

## Libraries to Work with Data

Exercises in this section are intended to be executed from an interactive Python interpreter.

[iPython](https://ipython.org) has been installed as part of the requirements.txt installation and is one option. You can start an iPython window by simply typing ipython. For each step in the list below, type the specified command (or commands) and then press enter until the iPython prompt goes to the next step, and/or shows the expected output (ie. print or pprint commands).

Other options could be just python or idle.

### XML with xmltodict

1. From the root of the python\_networking repository, change into the exercise directory.

cd data\_manipulation/xml

1. Import the xmltodict library

import xmltodict  
  
# Open the sample xml file and read it into variable  
  
with open("xml\_example.xml") as f:  
 xml\_example = f.read()

---------------------------------------------------------------------------  
  
FileNotFoundError Traceback (most recent call last)  
  
~\AppData\Local\Temp\ipykernel\_23500\2970058741.py in <module>  
 3 # 1. Open the sample xml file and read it into variable  
 4   
----> 5 with open("xml\_example.xml") as f:  
 6 xml\_example = f.read()  
  
  
FileNotFoundError: [Errno 2] No such file or directory: 'xml\_example.xml'

# Print the raw XML data  
  
print(xml\_example)

# Parse the XML into a Python (Ordered) dictionary  
  
xml\_dict = xmltodict.parse(xml\_example)

# Pretty Print the Python Dictionary Object  
  
from pprint import pprint  
  
pprint(xml\_dict)

# Save the interface name into a variable using XML nodes as keys  
  
int\_name = xml\_dict["interface"]["name"]

Print  
the  
interface  
name

print(int\_name)

Change  
the  
IP  
address  
of  
the  
interface

xml\_dict[“interface”][“ipv4”][“address”][“ip”] = “192.168.0.2”

Check  
that  
the  
IP  
address  
has  
been  
changed in the  
dictionary

pprint(xml\_dict)

Revert  
to  
the  
XML  
string  
version  
of  
the  
dictionary

print(xmltodict.unparse(xml\_dict))

### JSON with json  
From  
the  
root  
of  
the  
`python\_networking`  
repository, change  
into  
the  
exercise  
directory.

cd data\_manipulation/json

Import the jsontodict library

import json

Open  
the  
sample  
json  
file and read  
it  
into  
variable

with open(“json\_example.json”) as f: json\_example = f.read()

Print  
the  
raw  
json  
data

print(json\_example)

Parse  
the  
json  
into  
a  
Python  
dictionary

json\_dict = json.loads(json\_example)

Pretty  
Print  
the  
Python  
Dictionary  
Object

from pprint import pprint

pprint(json\_dict)

1.  
Save  
the  
interface  
name  
into  
a  
variable

int\_name = json\_dict[“interface”][“name”]

1.  
Print  
the  
interface  
name

print(int\_name)

1.  
Change  
the  
IP  
address  
of  
the  
interface

json\_dict[“interface”][“ipv4”][“address”][0][“ip”] = “192.168.0.2”

1.  
Check  
that  
the  
IP  
address  
has  
been  
changed in the  
dictionary

pprint(json\_dict)

1.  
Revert  
to  
the  
json  
string  
version  
of  
the  
dictionary

print(json.dumps(json\_dict))

### YAML with PyYAML  
1.  
From  
the  
root  
of  
the  
`python\_networking`  
repository, change  
into  
the  
exercise  
directory.

cd data\_manipulation/yaml

1. Import the yamltodict library

import yaml

1.  
Open  
the  
sample  
yaml  
file and read  
it  
into  
variable

with open(“yaml\_example.yaml”) as f: yaml\_example = f.read()

1.  
Print  
the  
raw  
yaml  
data

print(yaml\_example)

1.  
Parse  
the  
yaml  
into  
a  
Python  
dictionary

yaml\_dict = yaml.load(yaml\_example)

1. Pretty Print the Python Dictionary Object

from pprint import pprint  
  
pprint(yaml\_dict)

1. Save the interface name into a variable

int\_name = yaml\_dict["interface"]["name"]

1. Print the interface name

print(int\_name)

1. Change the IP address of the interface

yaml\_dict["interface"]["ipv4"]["address"][0]["ip"] = "192.168.0.2"

1. Check that the IP address has been changed in the dictionary

pprint(yaml\_dict)

1. Revert to the yaml string version of the dictionary

print(yaml.dump(yaml\_dict, default\_flow\_style=False))

### CSV with csv

1. From the root of the python\_networking repository, change into the exercise directory.

cd data\_manipulation/csv

1. Import the csv library

import csv

1. Open the sample csv file and print it to screen

with open("csv\_example.csv") as f:  
 print(f.read())

1. Open the sample csv file, and create a csv.reader object

with open("csv\_example.csv") as f:  
 csv\_python = csv.reader(f)  
 # Loop over each row in csv and leverage the data in code  
 for row in csv\_python:  
 print("{device} is in {location} "  
 "and has IP {ip}.".format(  
 device=row[0],  
 location=row[2],  
 ip=row[1]  
 )  
 )

1. Create a new tuple for additional router.

router4 = ("router4", "10.4.0.1", "Chicago")

1. Add new router to CSV file.

with open("csv\_example.csv", "a") as f:  
 csv\_writer = csv.writer(f)  
 csv\_writer.writerow(router4)

1. Re-read and print out the CSV content.

with open("csv\_example.csv") as f:  
 print(f.read())

### YANG with pyang

1. From the root of the python\_networking repository, change into the exercise directory.

cd data\_manipulation/yang

1. Print the YANG module in a simple text tree

pyang -f tree ietf-interfaces.yang

1. Print only part of the tree

pyang -f tree --tree-path=/interfaces/interface \  
 ietf-interfaces.yang

1. Print an example XML skeleton (NETCONF)

pyang -f sample-xml-skeleton ietf-interfaces.yang

1. Create an HTTP/JS view of the YANG Model (no output expected in the CLI)

pyang -f jstree -o ietf-interfaces.html \  
 ietf-interfaces.yang

1. *Optional:* Open ietf-interfaces.html in a web browser. Will need to RDP into the Devbox to do this step.
2. Control the “nested depth” in trees

pyang -f tree --tree-depth=2 ietf-ip.yang

1. Display a full module.

pyang -f tree \  
 ietf-ip.yang

1. Include deviation models in the processing

pyang -f tree \  
 --deviation-module=cisco-xe-ietf-ip-deviation.yang \  
 ietf-ip.yang

## Libraries to Work with APIs

Exercises in this section are intended to be executed from an interactive Python interpreter.

[iPython](https://ipython.org) has been installed as part of the requirements.txt installation and is one option. You can start an iPython window by simply typing ipython.

Other options could be just python or idle.

Each exercise also includes a Python script file that can be executed directly.

### REST with requests

1. From the root of the python\_networking repository, change into the exercise directory.

cd device\_apis/rest

### Retrieve Network Configuration Details with RESTCONF with restconf\_example1.py

1. Import libraries

import requests, urllib3  
import sys

1. Add parent directory to path to allow importing common vars

sys.path.append("..")  
  
from device\_info import vagrant\_iosxe as device

1. Disable Self-Signed Cert warning for demo

urllib3.disable\_warnings(urllib3.exceptions.InsecureRequestWarning)

1. Setup base variable for request

restconf\_headers = {"Accept": "application/yang-data+json"}  
restconf\_base = "https://{ip}:{port}/restconf/data"  
interface\_url = restconf\_base + "/ietf-interfaces:interfaces/interface={int\_name}"

1. Create URL GigE2 Config

url = interface\_url.format(ip=device["address"],  
 port=device["restconf\_port"],  
 int\_name="GigabitEthernet2"  
 )

1. Check the complete URL you just composed

print(url)

1. Send RESTCONF request to core1 for GigE2 Config

r = requests.get(url,  
 headers=restconf\_headers,  
 auth=(device["username"], device["password"]),  
 verify=False)

1. Print returned data

print(r.text)

1. If REST call was successful, report interesting details.

if r.status\_code == 200:  
 # Process JSON data into Python Dictionary and use  
 interface = r.json()["ietf-interfaces:interface"]  
 print("The interface {name} has ip address {ip}/{mask}".format(  
 name=interface["name"],  
 ip=interface["ietf-ip:ipv4"]["address"][0]["ip"],  
 mask=interface["ietf-ip:ipv4"]["address"][0]["netmask"],  
 )  
 )  
else:  
 print("No interface {} found.".format("GigabitEthernet2"))

### Modify Network Configuration Details with RESTCONF with restconf\_example2.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

import requests, urllib3, sys  
  
sys.path.append("..")  
  
from device\_info import vagrant\_iosxe as device  
  
urllib3.disable\_warnings(urllib3.exceptions.InsecureRequestWarning)  
restconf\_headers = {"Accept": "application/yang-data+json"}  
restconf\_base = "https://{ip}:{port}/restconf/data"  
interface\_url = restconf\_base + "/ietf-interfaces:interfaces/interface={int\_name}"

1. Add additional Content-Type header.

restconf\_headers["Content-Type"] = "application/yang-data+json"

1. Create dictionary with details on a new loopback interface.

loopback = {"name": "Loopback101",  
 "description": "Demo interface by RESTCONF",  
 "ip": "192.168.101.1",  
 "netmask": "255.255.255.0"}

1. Setup data body to create new loopback interface

data = {  
 "ietf-interfaces:interface": {  
 "name": loopback["name"],  
 "description": loopback["description"],  
 "type": "iana-if-type:softwareLoopback",  
 "enabled": True,  
 "ietf-ip:ipv4": {  
 "address": [  
 {  
 "ip": loopback["ip"],  
 "netmask": loopback["netmask"]  
 }  
 ]  
 }  
 }  
}

1. Create URL

url = interface\_url.format(ip=device["address"],  
 port=device["restconf\_port"],  
 int\_name=loopback["name"]  
 )

1. Check the complete URL you just composed

print(url)

1. Send RESTCONF request to device

r = requests.put(url,  
 headers=restconf\_headers,  
 auth=(device["username"], device["password"]),  
 json=data,  
 verify=False)

1. Check Status Code (expected 201)

print("Request Status Code: {}".format(r.status\_code))

1. Query for details on the new interface you just created

# Create URL and send RESTCONF request to core1 for GigE2 Config  
url = interface\_url.format(ip=device["address"],  
 port=device["restconf\_port"],  
 int\_name="Loopback101"  
 )  
r = requests.get(url,  
 headers=restconf\_headers,  
 auth=(device["username"], device["password"]),  
 verify=False)  
  
# Print returned data  
print(r.text)

### Delete Network Configuration Details with RESTCONF with restconf\_example3.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

import requests, urllib3, sys  
  
sys.path.append("..")  
from device\_info import vagrant\_iosxe as device  
  
urllib3.disable\_warnings(urllib3.exceptions.InsecureRequestWarning)  
restconf\_headers = {"Accept": "application/yang-data+json"}  
restconf\_base = "https://{ip}:{port}/restconf/data"  
interface\_url = restconf\_base + "/ietf-interfaces:interfaces/interface={int\_name}"  
url = interface\_url.format(ip=device["address"],  
 port=device["restconf\_port"],  
 int\_name="Loopback101"  
 )

1. Send DELETE request to remove the Loopback.

r = requests.delete(url,  
 headers=restconf\_headers,  
 auth=(device["username"], device["password"]),  
 verify=False)

1. Check Status Code (expected 204)

print("Request Status Code: {}".format(r.status\_code))

1. Query for details on the new interface (no output expected, as you just deleted it)

r = requests.get(url,  
 headers=restconf\_headers,  
 auth=(device["username"], device["password"]),  
 verify=False)

1. Check status code (expected 404)

print(r.status\_code)

## NETCONF with ncclient

1. From the root of the python\_networking repository, change into the exercise directory.

cd device\_apis/netconf

### Retrieve Network Configuration Details with NETCONF with netconf\_example1.py

1. Import libraries

from ncclient import manager  
from xml.dom import minidom  
import xmltodict  
import sys

1. Add parent directory to path to allow importing common vars

sys.path.append("..")  
  
from device\_info import vagrant\_iosxe as device

1. Create filter template for an interface

interface\_filter = """  
<filter>  
 <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
 <interface>  
 <name>{int\_name}</name>  
 </interface>  
 </interfaces>  
</filter>  
"""

1. Open NETCONF connection to device
   * *Note: Normally you’d use a with block to open connection to device. This avoids needing to manually m.close\_session() at the end of a script, but for interactive use, this format is chosen.*

m = manager.connect(host=device["address"],  
 port=device["netconf\_port"],  
 username=device["username"],  
 password=device["password"],  
 hostkey\_verify=False)

1. Verify NETCONF connection is active (expected output true)

m.connected

1. Create desired NETCONF filter for a particular interface

filter = interface\_filter.format(int\_name="GigabitEthernet2")

1. Execute a NETCONF using the filter

r = m.get\_config("running", filter)

1. Pretty print raw xml to screen

xml\_doc = minidom.parseString(r.xml)  
print(xml\_doc.toprettyxml(indent=" "))

1. Process the XML data into Python Dictionary and use

interface = xmltodict.parse(r.xml)

1. Pretty Print the full Python (Ordered) Dictionary.

from pprint import pprint  
  
pprint(interface)

1. If RPC returned data, print out the interesting pieces.

if not interface["rpc-reply"]["data"] is None:  
 # Create Python variable for interface details  
 interface = interface["rpc-reply"]["data"]["interfaces"]["interface"]  
  
 print("The interface {name} has ip address {ip}/{mask}".format(  
 name=interface["name"]["#text"],  
 ip=interface["ipv4"]["address"]["ip"],  
 mask=interface["ipv4"]["address"]["netmask"],  
 )  
 )  
else:  
 print("No interface {} found".format("GigabitEthernet2"))

### Modify Network Configuration Details with NETCONF with netconf\_example2.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

from ncclient import manager  
from xml.dom import minidom  
import xmltodict  
import sys  
  
sys.path.append("..")  
  
from device\_info import vagrant\_iosxe as device  
  
interface\_filter = """  
<filter>  
 <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
 <interface>  
 <name>{int\_name}</name>  
 </interface>  
 </interfaces>  
</filter>  
"""  
  
m = manager.connect(host=device["address"],  
 port=device["netconf\_port"],  
 username=device["username"],  
 password=device["password"],  
 hostkey\_verify=False)

1. Verify NETCONF connection is active

m.connected

1. Create Python dictionary with new Loopback Details

loopback = {"int\_name": "Loopback102",  
 "description": "Demo interface by NETCONF",  
 "ip": "192.168.102.1",  
 "netmask": "255.255.255.0"}

1. Create NETCONF template for an interface

config\_data = """  
<config>  
 <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
 <interface>  
 <name>{int\_name}</name>  
 <description>{description}</description>  
 <type xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">  
 ianaift:softwareLoopback  
 </type>  
 <enabled>true</enabled>  
 <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">  
 <address>  
 <ip>{ip}</ip>  
 <netmask>{netmask}</netmask>  
 </address>  
 </ipv4>  
 </interface>  
 </interfaces>  
</config>  
"""

1. Create desired NETCONF config payload

config = config\_data.format(\*\*loopback)

1. Send operation

r = m.edit\_config(target="running", config=config)

1. Print OK status (expected output true)

print("NETCONF RPC OK: {}".format(r.ok))

1. Create a new NETCONF to check on new loopback interface

filter = interface\_filter.format(int\_name="Loopback102")

1. Execute a NETCONF using this filter

r = m.get\_config("running", filter)

1. Pretty print the raw XML to screen

xml\_doc = minidom.parseString(r.xml)  
print(xml\_doc.toprettyxml(indent=" "))

### Delete Network Configuration Details with NETCONF with netconf\_example3.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

from ncclient import manager  
from xml.dom import minidom  
import xmltodict  
import sys  
  
sys.path.append("..")  
  
from device\_info import vagrant\_iosxe as device  
  
interface\_filter = """  
<filter>  
 <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
 <interface>  
 <name>{int\_name}</name>  
 </interface>  
 </interfaces>  
</filter>  
"""  
loopback = {"int\_name": "Loopback102",  
 "description": "Demo interface by NETCONF",  
 "ip": "192.168.102.1",  
 "netmask": "255.255.255.0"}  
m = manager.connect(host=device["address"],  
 port=device["netconf\_port"],  
 username=device["username"],  
 password=device["password"],  
 hostkey\_verify=False)

1. Verify NETCONF connection is active

m.connected

1. Create new config template to delete an interface

config\_data = """  
<config>  
 <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
 <interface operation="delete">  
 <name>{int\_name}</name>  
 </interface>  
 </interfaces>  
</config>  
"""

1. Create desired NETCONF config payload and execute to delete the interface

config = config\_data.format(\*\*loopback)  
r = m.edit\_config(target="running", config=config)

1. Print OK status (expected output true)

print("NETCONF RPC OK: {}".format(r.ok))

1. Create a new NETCONF to check on new loopback interface

filter = interface\_filter.format(int\_name="Loopback102")

1. Execute a NETCONF using this filter

r = m.get\_config("running", filter)

1. Pretty print the raw XML to screen (expected output will not include the loopback interface, as you just deleted it)

xml\_doc = minidom.parseString(r.xml)  
print(xml\_doc.toprettyxml(indent=" "))

### End the NETCONF Connection

1. Send a RPC request to disconnect the connection.

m.close\_session()  
m.connected

## CLI with netmiko

1. From the root of the python\_networking repository, change into the exercise directory.

cd device\_apis/cli

### Retrieve Network Configuration Details with CLI with netmiko\_example1.py

1. Import libraries

from netmiko import ConnectHandler  
import re  
import sys

1. Add parent directory to path to allow importing common vars

sys.path.append("..")  
from device\_info import vagrant\_iosxe as device

1. Set device\_type for netmiko

device["device\_type"] = "cisco\_ios"

1. Create a CLI command template

show\_interface\_config\_temp = "show running-config interface {}"

1. Open CLI connection to device.
   * *Note: Normally you’d use a with block to open connection to device. This avoids needing to manually m.close\_session() at the end of a script, but for interactive use, this format is chosen.*

ch = ConnectHandler(ip=device["address"],  
 port=device["ssh\_port"],  
 username=device["username"],  
 password=device["password"],  
 device\_type=device["device\_type"])

1. Create desired CLI command

command = show\_interface\_config\_temp.format("GigabitEthernet2")

1. Verify the command has been created correctly

print(command)

1. Send command to device

interface = ch.send\_command(command)

1. Print the raw command output to the screen

print(interface)

1. Create regular expression searches to parse the output for desired interface details

name = re.search(r'interface (.\*)', interface).group(1)  
description = re.search(r'description (.\*)', interface).group(1)

1. Pull out the ip and mask for the interface

ip\_info = re.search(r'ip address (.\*) (.\*)', interface)  
ip = ip\_info.group(1)  
netmask = ip\_info.group(2)

1. Print the desired info to the screen

print("The interface {name} has ip address {ip}/{mask}".format(  
 name=name,  
 ip=ip,  
 mask=netmask,  
)  
)

### Modify Network Configuration Details with CLI with netmiko\_example2.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

from netmiko import ConnectHandler  
import re, sys  
  
sys.path.append("..")  
from device\_info import vagrant\_iosxe as device  
  
device["device\_type"] = "cisco\_ios"  
show\_interface\_config\_temp = "show running-config interface {}"  
ch = ConnectHandler(ip=device["address"],  
 port=device["ssh\_port"],  
 username=device["username"],  
 password=device["password"],  
 device\_type=device["device\_type"])

1. Create Python dictionary with new Loopback Details

loopback = {"int\_name": "Loopback103",  
 "description": "Demo interface by CLI and netmiko",  
 "ip": "192.168.103.1",  
 "netmask": "255.255.255.0"}

1. Create a CLI configuration

interface\_config = [  
 "interface {}".format(loopback["int\_name"]),  
 "description {}".format(loopback["description"]),  
 "ip address {} {}".format(loopback["ip"], loopback["netmask"]),  
 "no shut"  
]

1. Send configuration to device

output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

print("The following configuration was sent: ")  
print(output)

1. Create a CLI command to retrieve the new configuration.

command = show\_interface\_config\_temp.format("Loopback103")  
interface = ch.send\_command(command)  
print(interface)

### Delete Network Configuration Details with CLI with netmiko\_example3.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

from netmiko import ConnectHandler  
import re, sys  
  
sys.path.append("..")  
  
from device\_info import vagrant\_iosxe as device  
  
device["device\_type"] = "cisco\_ios"  
show\_interface\_config\_temp = "show running-config interface {}"  
ch = ConnectHandler(ip=device["address"],  
 port=device["ssh\_port"],  
 username=device["username"],  
 password=device["password"],  
 device\_type=device["device\_type"])

1. Create a new CLI configuration to delete the interface.

interface\_config = [  
 "no interface {}".format(loopback["int\_name"])  
]

1. Send configuration to device

output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

print(“The following configuration was sent:”) print(output)

1. Create a CLI command to verify configuration removed.

command = show\_interface\_config\_temp.format("Loopback103")  
interface = ch.send\_command(command)  
print(interface)

**Note**: attempting to view the configuration of a non-existing interface will generate a CLI error. This output is expected, and one of the reasons APIs like NETCONF or RESTCONF are better suited to programmatic interactions.

### End the CLI connection to the device

1. Disconnect from the device.

ch.disconnect()

## Other Cool Python Stuff

### Introduction to pyATS

[pyATS](https://developer.cisco.com/site/pyats) is a network testing tool developed by Cisco and made available for free, with significant elements of the underlying code open source.

**pyATS** offers network developers the ability to profile the network state of hardware, interfaces, protocols, etc… before, during and after changes, to ensure the network is operating as designed, and identify problems before the dreaded phone call. To enable this level of robust testing, pyATS offers a standard way to communicate with network elements and standardize the data returned into native Python objects. This core functionality opens up a lot of flexibility on how pyATS can be used by network developers.

In the following exercises, you will get a brief introduction to pyATS to connect and learn about device details.

### Connect and Interact with a Device

1. From the root of the python\_networking repository, change into the exercise directory.

cd network\_testing/pyats

1. Import in pyATS libraries and tools

from genie.conf import Genie  
from ats.topology import loader  
from genie.abstract import Lookup  
from genie.libs import ops # noqa

1. Read and process the testbed (inventory) file

genie\_testbed = Genie.init("./default\_testbed.yaml")

1. Create a pyATS device object from testbed

vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

vagrant\_iosxe1.connect()

* pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

config\_loopback = [  
 "interface Loopback201",  
 "description Configured by pyATS",  
 "ip address 172.16.201.1 255.255.255.0",  
 "no shut"  
]  
vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

vagrant\_iosxe1.disconnect()

```python  
config\_loopback = [  
 "interface Loopback201",  
 "description Configured by pyATS",  
 "ip address 172.16.201.1 255.255.255.0",  
 "no shut"  
 ]  
vagrant\_iosxe1.configure(config\_loopback)  
```

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()
* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()
* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()
* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()
* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()
* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()
* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()
* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()
* vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()

1. Create a pyATS device object from testbed

* vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()
* genie\_testbed = Genie.init("./default\_testbed.yaml")

1. Create a pyATS device object from testbed

* vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()

1. Read and process the testbed (inventory) file

* genie\_testbed = Genie.init("./default\_testbed.yaml")

1. Create a pyATS device object from testbed

* vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()

1. Import in pyATS libraries and tools
2. Import in pyATS libraries and tools

* from genie.conf import Genie  
  from ats.topology import loader  
  from genie.abstract import Lookup  
  from genie.libs import ops # noqa

1. Read and process the testbed (inventory) file

* genie\_testbed = Genie.init("./default\_testbed.yaml")

1. Create a pyATS device object from testbed

* vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()
* from genie.conf import Genie  
  from ats.topology import loader  
  from genie.abstract import Lookup  
  from genie.libs import ops # noqa

1. Read and process the testbed (inventory) file

* genie\_testbed = Genie.init("./default\_testbed.yaml")

1. Create a pyATS device object from testbed

* vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()

1. Import in pyATS libraries and tools

* from genie.conf import Genie  
  from ats.topology import loader  
  from genie.abstract import Lookup  
  from genie.libs import ops # noqa

1. Read and process the testbed (inventory) file

* genie\_testbed = Genie.init("./default\_testbed.yaml")

1. Create a pyATS device object from testbed

* vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()

## Other Cool Python Stuff

### Introduction to pyATS

[pyATS](https://developer.cisco.com/site/pyats) is a network testing tool developed by Cisco and made available for free, with significant elements of the underlying code open source.

pyATS offers network developers the ability to profile the network state of hardware, interfaces, protocols, etc… before, during and after changes, to ensure the network is operating as designed, and identify problems before the dreaded phone call. To enable this level of robust testing, pyATS offers a standard way to communicate with network elements and standardize the data returned into native Python objects. This core functionality opens up a lot of flexibility on how pyATS can be used by network developers.

In the following exercises, you will get a brief introduction to pyATS to connect and learn about device details.

### Connect and Interact with a Device

1. From the root of the python\_networking repository, change into the exercise directory.

* cd network\_testing/pyats

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

1. Import in pyATS libraries and tools

* from genie.conf import Genie  
  from ats.topology import loader  
  from genie.abstract import Lookup  
  from genie.libs import ops # noqa

1. Read and process the testbed (inventory) file

* genie\_testbed = Genie.init("./default\_testbed.yaml")

1. Create a pyATS device object from testbed

* vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()
* exit()

## Other Cool Python Stuff

### Introduction to pyATS

[pyATS](https://developer.cisco.com/site/pyats) is a network testing tool developed by Cisco and made available for free, with significant elements of the underlying code open source.

pyATS offers network developers the ability to profile the network state of hardware, interfaces, protocols, etc… before, during and after changes, to ensure the network is operating as designed, and identify problems before the dreaded phone call. To enable this level of robust testing, pyATS offers a standard way to communicate with network elements and standardize the data returned into native Python objects. This core functionality opens up a lot of flexibility on how pyATS can be used by network developers.

In the following exercises, you will get a brief introduction to pyATS to connect and learn about device details.

### Connect and Interact with a Device

1. From the root of the python\_networking repository, change into the exercise directory.

* cd network\_testing/pyats

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

1. Import in pyATS libraries and tools

* from genie.conf import Genie  
  from ats.topology import loader  
  from genie.abstract import Lookup  
  from genie.libs import ops # noqa

1. Read and process the testbed (inventory) file

* genie\_testbed = Genie.init("./default\_testbed.yaml")

1. Create a pyATS device object from testbed

* vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()

1. End the Python interpreter.

* exit()

## Other Cool Python Stuff

### Introduction to pyATS

[pyATS](https://developer.cisco.com/site/pyats) is a network testing tool developed by Cisco and made available for free, with significant elements of the underlying code open source.

pyATS offers network developers the ability to profile the network state of hardware, interfaces, protocols, etc… before, during and after changes, to ensure the network is operating as designed, and identify problems before the dreaded phone call. To enable this level of robust testing, pyATS offers a standard way to communicate with network elements and standardize the data returned into native Python objects. This core functionality opens up a lot of flexibility on how pyATS can be used by network developers.

In the following exercises, you will get a brief introduction to pyATS to connect and learn about device details.

### Connect and Interact with a Device

1. From the root of the python\_networking repository, change into the exercise directory.

* cd network\_testing/pyats

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

1. Import in pyATS libraries and tools

* from genie.conf import Genie  
  from ats.topology import loader  
  from genie.abstract import Lookup  
  from genie.libs import ops # noqa

1. Read and process the testbed (inventory) file

* genie\_testbed = Genie.init("./default\_testbed.yaml")

1. Create a pyATS device object from testbed

* vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()
* ch.disconnect()

1. End the Python interpreter.

* exit()

## Other Cool Python Stuff

### Introduction to pyATS

[pyATS](https://developer.cisco.com/site/pyats) is a network testing tool developed by Cisco and made available for free, with significant elements of the underlying code open source.

pyATS offers network developers the ability to profile the network state of hardware, interfaces, protocols, etc… before, during and after changes, to ensure the network is operating as designed, and identify problems before the dreaded phone call. To enable this level of robust testing, pyATS offers a standard way to communicate with network elements and standardize the data returned into native Python objects. This core functionality opens up a lot of flexibility on how pyATS can be used by network developers.

In the following exercises, you will get a brief introduction to pyATS to connect and learn about device details.

### Connect and Interact with a Device

1. From the root of the python\_networking repository, change into the exercise directory.

* cd network\_testing/pyats

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

1. Import in pyATS libraries and tools

* from genie.conf import Genie  
  from ats.topology import loader  
  from genie.abstract import Lookup  
  from genie.libs import ops # noqa

1. Read and process the testbed (inventory) file

* genie\_testbed = Genie.init("./default\_testbed.yaml")

1. Create a pyATS device object from testbed

* vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()
* *Note: attempting to view the configuration of a non-existing interface will generate a CLI error. This output is expected, and one of the reasons APIs like NETCONF or RESTCONF are better suited to programmatic interactions.*

### End the CLI connection to the device

1. Disconnect from the device.

* ch.disconnect()

1. End the Python interpreter.

* exit()

## Other Cool Python Stuff

### Introduction to pyATS

[pyATS](https://developer.cisco.com/site/pyats) is a network testing tool developed by Cisco and made available for free, with significant elements of the underlying code open source.

pyATS offers network developers the ability to profile the network state of hardware, interfaces, protocols, etc… before, during and after changes, to ensure the network is operating as designed, and identify problems before the dreaded phone call. To enable this level of robust testing, pyATS offers a standard way to communicate with network elements and standardize the data returned into native Python objects. This core functionality opens up a lot of flexibility on how pyATS can be used by network developers.

In the following exercises, you will get a brief introduction to pyATS to connect and learn about device details.

### Connect and Interact with a Device

1. From the root of the python\_networking repository, change into the exercise directory.

* cd network\_testing/pyats

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

1. Import in pyATS libraries and tools

* from genie.conf import Genie  
  from ats.topology import loader  
  from genie.abstract import Lookup  
  from genie.libs import ops # noqa

1. Read and process the testbed (inventory) file

* genie\_testbed = Genie.init("./default\_testbed.yaml")

1. Create a pyATS device object from testbed

* vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()
* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)
* *Note: attempting to view the configuration of a non-existing interface will generate a CLI error. This output is expected, and one of the reasons APIs like NETCONF or RESTCONF are better suited to programmatic interactions.*

### End the CLI connection to the device

1. Disconnect from the device.

* ch.disconnect()

1. End the Python interpreter.

* exit()

## Other Cool Python Stuff

### Introduction to pyATS

[pyATS](https://developer.cisco.com/site/pyats) is a network testing tool developed by Cisco and made available for free, with significant elements of the underlying code open source.

pyATS offers network developers the ability to profile the network state of hardware, interfaces, protocols, etc… before, during and after changes, to ensure the network is operating as designed, and identify problems before the dreaded phone call. To enable this level of robust testing, pyATS offers a standard way to communicate with network elements and standardize the data returned into native Python objects. This core functionality opens up a lot of flexibility on how pyATS can be used by network developers.

In the following exercises, you will get a brief introduction to pyATS to connect and learn about device details.

### Connect and Interact with a Device

1. From the root of the python\_networking repository, change into the exercise directory.

* cd network\_testing/pyats

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

1. Import in pyATS libraries and tools

* from genie.conf import Genie  
  from ats.topology import loader  
  from genie.abstract import Lookup  
  from genie.libs import ops # noqa

1. Read and process the testbed (inventory) file

* genie\_testbed = Genie.init("./default\_testbed.yaml")

1. Create a pyATS device object from testbed

* vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()

1. Create a CLI command to verify configuration removed.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)
* *Note: attempting to view the configuration of a non-existing interface will generate a CLI error. This output is expected, and one of the reasons APIs like NETCONF or RESTCONF are better suited to programmatic interactions.*

### End the CLI connection to the device

1. Disconnect from the device.

* ch.disconnect()

1. End the Python interpreter.

* exit()

## Other Cool Python Stuff

### Introduction to pyATS

[pyATS](https://developer.cisco.com/site/pyats) is a network testing tool developed by Cisco and made available for free, with significant elements of the underlying code open source.

pyATS offers network developers the ability to profile the network state of hardware, interfaces, protocols, etc… before, during and after changes, to ensure the network is operating as designed, and identify problems before the dreaded phone call. To enable this level of robust testing, pyATS offers a standard way to communicate with network elements and standardize the data returned into native Python objects. This core functionality opens up a lot of flexibility on how pyATS can be used by network developers.

In the following exercises, you will get a brief introduction to pyATS to connect and learn about device details.

### Connect and Interact with a Device

1. From the root of the python\_networking repository, change into the exercise directory.

* cd network\_testing/pyats

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

1. Import in pyATS libraries and tools

* from genie.conf import Genie  
  from ats.topology import loader  
  from genie.abstract import Lookup  
  from genie.libs import ops # noqa

1. Read and process the testbed (inventory) file

* genie\_testbed = Genie.init("./default\_testbed.yaml")

1. Create a pyATS device object from testbed

* vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()
* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to verify configuration removed.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)
* *Note: attempting to view the configuration of a non-existing interface will generate a CLI error. This output is expected, and one of the reasons APIs like NETCONF or RESTCONF are better suited to programmatic interactions.*

### End the CLI connection to the device

1. Disconnect from the device.

* ch.disconnect()

1. End the Python interpreter.

* exit()

## Other Cool Python Stuff

### Introduction to pyATS

[pyATS](https://developer.cisco.com/site/pyats) is a network testing tool developed by Cisco and made available for free, with significant elements of the underlying code open source.

pyATS offers network developers the ability to profile the network state of hardware, interfaces, protocols, etc… before, during and after changes, to ensure the network is operating as designed, and identify problems before the dreaded phone call. To enable this level of robust testing, pyATS offers a standard way to communicate with network elements and standardize the data returned into native Python objects. This core functionality opens up a lot of flexibility on how pyATS can be used by network developers.

In the following exercises, you will get a brief introduction to pyATS to connect and learn about device details.

### Connect and Interact with a Device

1. From the root of the python\_networking repository, change into the exercise directory.

* cd network\_testing/pyats

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

1. Import in pyATS libraries and tools

* from genie.conf import Genie  
  from ats.topology import loader  
  from genie.abstract import Lookup  
  from genie.libs import ops # noqa

1. Read and process the testbed (inventory) file

* genie\_testbed = Genie.init("./default\_testbed.yaml")

1. Create a pyATS device object from testbed

* vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()

1. Send configuration to device

* output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to verify configuration removed.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)
* *Note: attempting to view the configuration of a non-existing interface will generate a CLI error. This output is expected, and one of the reasons APIs like NETCONF or RESTCONF are better suited to programmatic interactions.*

### End the CLI connection to the device

1. Disconnect from the device.

* ch.disconnect()

1. End the Python interpreter.

* exit()

## Other Cool Python Stuff

### Introduction to pyATS

[pyATS](https://developer.cisco.com/site/pyats) is a network testing tool developed by Cisco and made available for free, with significant elements of the underlying code open source.

pyATS offers network developers the ability to profile the network state of hardware, interfaces, protocols, etc… before, during and after changes, to ensure the network is operating as designed, and identify problems before the dreaded phone call. To enable this level of robust testing, pyATS offers a standard way to communicate with network elements and standardize the data returned into native Python objects. This core functionality opens up a lot of flexibility on how pyATS can be used by network developers.

In the following exercises, you will get a brief introduction to pyATS to connect and learn about device details.

### Connect and Interact with a Device

1. From the root of the python\_networking repository, change into the exercise directory.

* cd network\_testing/pyats

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

1. Import in pyATS libraries and tools

* from genie.conf import Genie  
  from ats.topology import loader  
  from genie.abstract import Lookup  
  from genie.libs import ops # noqa

1. Read and process the testbed (inventory) file

* genie\_testbed = Genie.init("./default\_testbed.yaml")

1. Create a pyATS device object from testbed

* vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()
* interface\_config = [  
   "no interface {}".format(loopback["int\_name"])  
  ]

1. Send configuration to device

* output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to verify configuration removed.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)
* *Note: attempting to view the configuration of a non-existing interface will generate a CLI error. This output is expected, and one of the reasons APIs like NETCONF or RESTCONF are better suited to programmatic interactions.*

### End the CLI connection to the device

1. Disconnect from the device.

* ch.disconnect()

1. End the Python interpreter.

* exit()

## Other Cool Python Stuff

### Introduction to pyATS

[pyATS](https://developer.cisco.com/site/pyats) is a network testing tool developed by Cisco and made available for free, with significant elements of the underlying code open source.

pyATS offers network developers the ability to profile the network state of hardware, interfaces, protocols, etc… before, during and after changes, to ensure the network is operating as designed, and identify problems before the dreaded phone call. To enable this level of robust testing, pyATS offers a standard way to communicate with network elements and standardize the data returned into native Python objects. This core functionality opens up a lot of flexibility on how pyATS can be used by network developers.

In the following exercises, you will get a brief introduction to pyATS to connect and learn about device details.

### Connect and Interact with a Device

1. From the root of the python\_networking repository, change into the exercise directory.

* cd network\_testing/pyats

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

1. Import in pyATS libraries and tools

* from genie.conf import Genie  
  from ats.topology import loader  
  from genie.abstract import Lookup  
  from genie.libs import ops # noqa

1. Read and process the testbed (inventory) file

* genie\_testbed = Genie.init("./default\_testbed.yaml")

1. Create a pyATS device object from testbed

* vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()

1. Create a new CLI configuration to delete the interface.

* interface\_config = [  
   "no interface {}".format(loopback["int\_name"])  
  ]

1. Send configuration to device

* output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to verify configuration removed.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)
* *Note: attempting to view the configuration of a non-existing interface will generate a CLI error. This output is expected, and one of the reasons APIs like NETCONF or RESTCONF are better suited to programmatic interactions.*

### End the CLI connection to the device

1. Disconnect from the device.

* ch.disconnect()

1. End the Python interpreter.

* exit()

## Other Cool Python Stuff

### Introduction to pyATS

[pyATS](https://developer.cisco.com/site/pyats) is a network testing tool developed by Cisco and made available for free, with significant elements of the underlying code open source.

pyATS offers network developers the ability to profile the network state of hardware, interfaces, protocols, etc… before, during and after changes, to ensure the network is operating as designed, and identify problems before the dreaded phone call. To enable this level of robust testing, pyATS offers a standard way to communicate with network elements and standardize the data returned into native Python objects. This core functionality opens up a lot of flexibility on how pyATS can be used by network developers.

In the following exercises, you will get a brief introduction to pyATS to connect and learn about device details.

### Connect and Interact with a Device

1. From the root of the python\_networking repository, change into the exercise directory.

* cd network\_testing/pyats

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

1. Import in pyATS libraries and tools

* from genie.conf import Genie  
  from ats.topology import loader  
  from genie.abstract import Lookup  
  from genie.libs import ops # noqa

1. Read and process the testbed (inventory) file

* genie\_testbed = Genie.init("./default\_testbed.yaml")

1. Create a pyATS device object from testbed

* vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()
* from netmiko import ConnectHandler  
  import re, sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  device["device\_type"] = "cisco\_ios"  
  show\_interface\_config\_temp = "show running-config interface {}"  
  ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create a new CLI configuration to delete the interface.

* interface\_config = [  
   "no interface {}".format(loopback["int\_name"])  
  ]

1. Send configuration to device

* output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to verify configuration removed.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)
* *Note: attempting to view the configuration of a non-existing interface will generate a CLI error. This output is expected, and one of the reasons APIs like NETCONF or RESTCONF are better suited to programmatic interactions.*

### End the CLI connection to the device

1. Disconnect from the device.

* ch.disconnect()

1. End the Python interpreter.

* exit()

## Other Cool Python Stuff

### Introduction to pyATS

[pyATS](https://developer.cisco.com/site/pyats) is a network testing tool developed by Cisco and made available for free, with significant elements of the underlying code open source.

pyATS offers network developers the ability to profile the network state of hardware, interfaces, protocols, etc… before, during and after changes, to ensure the network is operating as designed, and identify problems before the dreaded phone call. To enable this level of robust testing, pyATS offers a standard way to communicate with network elements and standardize the data returned into native Python objects. This core functionality opens up a lot of flexibility on how pyATS can be used by network developers.

In the following exercises, you will get a brief introduction to pyATS to connect and learn about device details.

### Connect and Interact with a Device

1. From the root of the python\_networking repository, change into the exercise directory.

* cd network\_testing/pyats

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

1. Import in pyATS libraries and tools

* from genie.conf import Genie  
  from ats.topology import loader  
  from genie.abstract import Lookup  
  from genie.libs import ops # noqa

1. Read and process the testbed (inventory) file

* genie\_testbed = Genie.init("./default\_testbed.yaml")

1. Create a pyATS device object from testbed

* vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()

### Delete Network Configuration Details with CLI with netmiko\_example3.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from netmiko import ConnectHandler  
  import re, sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  device["device\_type"] = "cisco\_ios"  
  show\_interface\_config\_temp = "show running-config interface {}"  
  ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create a new CLI configuration to delete the interface.

* interface\_config = [  
   "no interface {}".format(loopback["int\_name"])  
  ]

1. Send configuration to device

* output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to verify configuration removed.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)
* *Note: attempting to view the configuration of a non-existing interface will generate a CLI error. This output is expected, and one of the reasons APIs like NETCONF or RESTCONF are better suited to programmatic interactions.*

### End the CLI connection to the device

1. Disconnect from the device.

* ch.disconnect()

1. End the Python interpreter.

* exit()

## Other Cool Python Stuff

### Introduction to pyATS

[pyATS](https://developer.cisco.com/site/pyats) is a network testing tool developed by Cisco and made available for free, with significant elements of the underlying code open source.

pyATS offers network developers the ability to profile the network state of hardware, interfaces, protocols, etc… before, during and after changes, to ensure the network is operating as designed, and identify problems before the dreaded phone call. To enable this level of robust testing, pyATS offers a standard way to communicate with network elements and standardize the data returned into native Python objects. This core functionality opens up a lot of flexibility on how pyATS can be used by network developers.

In the following exercises, you will get a brief introduction to pyATS to connect and learn about device details.

### Connect and Interact with a Device

1. From the root of the python\_networking repository, change into the exercise directory.

* cd network\_testing/pyats

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

1. Import in pyATS libraries and tools

* from genie.conf import Genie  
  from ats.topology import loader  
  from genie.abstract import Lookup  
  from genie.libs import ops # noqa

1. Read and process the testbed (inventory) file

* genie\_testbed = Genie.init("./default\_testbed.yaml")

1. Create a pyATS device object from testbed

* vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()
* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)

### Delete Network Configuration Details with CLI with netmiko\_example3.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from netmiko import ConnectHandler  
  import re, sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  device["device\_type"] = "cisco\_ios"  
  show\_interface\_config\_temp = "show running-config interface {}"  
  ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create a new CLI configuration to delete the interface.

* interface\_config = [  
   "no interface {}".format(loopback["int\_name"])  
  ]

1. Send configuration to device

* output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to verify configuration removed.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)
* *Note: attempting to view the configuration of a non-existing interface will generate a CLI error. This output is expected, and one of the reasons APIs like NETCONF or RESTCONF are better suited to programmatic interactions.*

### End the CLI connection to the device

1. Disconnect from the device.

* ch.disconnect()

1. End the Python interpreter.

* exit()

## Other Cool Python Stuff

### Introduction to pyATS

[pyATS](https://developer.cisco.com/site/pyats) is a network testing tool developed by Cisco and made available for free, with significant elements of the underlying code open source.

pyATS offers network developers the ability to profile the network state of hardware, interfaces, protocols, etc… before, during and after changes, to ensure the network is operating as designed, and identify problems before the dreaded phone call. To enable this level of robust testing, pyATS offers a standard way to communicate with network elements and standardize the data returned into native Python objects. This core functionality opens up a lot of flexibility on how pyATS can be used by network developers.

In the following exercises, you will get a brief introduction to pyATS to connect and learn about device details.

### Connect and Interact with a Device

1. From the root of the python\_networking repository, change into the exercise directory.

* cd network\_testing/pyats

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

1. Import in pyATS libraries and tools

* from genie.conf import Genie  
  from ats.topology import loader  
  from genie.abstract import Lookup  
  from genie.libs import ops # noqa

1. Read and process the testbed (inventory) file

* genie\_testbed = Genie.init("./default\_testbed.yaml")

1. Create a pyATS device object from testbed

* vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()

1. Print the raw command output to the screen

* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to retrieve the new configuration.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)

### Delete Network Configuration Details with CLI with netmiko\_example3.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from netmiko import ConnectHandler  
  import re, sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  device["device\_type"] = "cisco\_ios"  
  show\_interface\_config\_temp = "show running-config interface {}"  
  ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create a new CLI configuration to delete the interface.

* interface\_config = [  
   "no interface {}".format(loopback["int\_name"])  
  ]

1. Send configuration to device

* output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to verify configuration removed.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)
* *Note: attempting to view the configuration of a non-existing interface will generate a CLI error. This output is expected, and one of the reasons APIs like NETCONF or RESTCONF are better suited to programmatic interactions.*

### End the CLI connection to the device

1. Disconnect from the device.

* ch.disconnect()

1. End the Python interpreter.

* exit()

## Other Cool Python Stuff

### Introduction to pyATS

[pyATS](https://developer.cisco.com/site/pyats) is a network testing tool developed by Cisco and made available for free, with significant elements of the underlying code open source.

pyATS offers network developers the ability to profile the network state of hardware, interfaces, protocols, etc… before, during and after changes, to ensure the network is operating as designed, and identify problems before the dreaded phone call. To enable this level of robust testing, pyATS offers a standard way to communicate with network elements and standardize the data returned into native Python objects. This core functionality opens up a lot of flexibility on how pyATS can be used by network developers.

In the following exercises, you will get a brief introduction to pyATS to connect and learn about device details.

### Connect and Interact with a Device

1. From the root of the python\_networking repository, change into the exercise directory.

* cd network\_testing/pyats

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

1. Import in pyATS libraries and tools

* from genie.conf import Genie  
  from ats.topology import loader  
  from genie.abstract import Lookup  
  from genie.libs import ops # noqa

1. Read and process the testbed (inventory) file

* genie\_testbed = Genie.init("./default\_testbed.yaml")

1. Create a pyATS device object from testbed

* vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()
* output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to retrieve the new configuration.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)

### Delete Network Configuration Details with CLI with netmiko\_example3.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from netmiko import ConnectHandler  
  import re, sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  device["device\_type"] = "cisco\_ios"  
  show\_interface\_config\_temp = "show running-config interface {}"  
  ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create a new CLI configuration to delete the interface.

* interface\_config = [  
   "no interface {}".format(loopback["int\_name"])  
  ]

1. Send configuration to device

* output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to verify configuration removed.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)
* *Note: attempting to view the configuration of a non-existing interface will generate a CLI error. This output is expected, and one of the reasons APIs like NETCONF or RESTCONF are better suited to programmatic interactions.*

### End the CLI connection to the device

1. Disconnect from the device.

* ch.disconnect()

1. End the Python interpreter.

* exit()

## Other Cool Python Stuff

### Introduction to pyATS

[pyATS](https://developer.cisco.com/site/pyats) is a network testing tool developed by Cisco and made available for free, with significant elements of the underlying code open source.

pyATS offers network developers the ability to profile the network state of hardware, interfaces, protocols, etc… before, during and after changes, to ensure the network is operating as designed, and identify problems before the dreaded phone call. To enable this level of robust testing, pyATS offers a standard way to communicate with network elements and standardize the data returned into native Python objects. This core functionality opens up a lot of flexibility on how pyATS can be used by network developers.

In the following exercises, you will get a brief introduction to pyATS to connect and learn about device details.

### Connect and Interact with a Device

1. From the root of the python\_networking repository, change into the exercise directory.

* cd network\_testing/pyats

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

1. Import in pyATS libraries and tools

* from genie.conf import Genie  
  from ats.topology import loader  
  from genie.abstract import Lookup  
  from genie.libs import ops # noqa

1. Read and process the testbed (inventory) file

* genie\_testbed = Genie.init("./default\_testbed.yaml")

1. Create a pyATS device object from testbed

* vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()
* from netmiko import ConnectHandler  
  import re, sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  device["device\_type"] = "cisco\_ios"  
  show\_interface\_config\_temp = "show running-config interface {}"  
  ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create Python dictionary with new Loopback Details

* loopback = {"int\_name": "Loopback103",  
   "description": "Demo interface by CLI and netmiko",  
   "ip": "192.168.103.1",  
   "netmask": "255.255.255.0"}

1. Create a CLI configuration

* interface\_config = [  
   "interface {}".format(loopback["int\_name"]),  
   "description {}".format(loopback["description"]),  
   "ip address {} {}".format(loopback["ip"], loopback["netmask"]),  
   "no shut"  
  ]

1. Send configuration to device

* output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to retrieve the new configuration.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)

### Delete Network Configuration Details with CLI with netmiko\_example3.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from netmiko import ConnectHandler  
  import re, sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  device["device\_type"] = "cisco\_ios"  
  show\_interface\_config\_temp = "show running-config interface {}"  
  ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create a new CLI configuration to delete the interface.

* interface\_config = [  
   "no interface {}".format(loopback["int\_name"])  
  ]

1. Send configuration to device

* output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to verify configuration removed.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)
* *Note: attempting to view the configuration of a non-existing interface will generate a CLI error. This output is expected, and one of the reasons APIs like NETCONF or RESTCONF are better suited to programmatic interactions.*

### End the CLI connection to the device

1. Disconnect from the device.

* ch.disconnect()

1. End the Python interpreter.

* exit()

## Other Cool Python Stuff

### Introduction to pyATS

[pyATS](https://developer.cisco.com/site/pyats) is a network testing tool developed by Cisco and made available for free, with significant elements of the underlying code open source.

pyATS offers network developers the ability to profile the network state of hardware, interfaces, protocols, etc… before, during and after changes, to ensure the network is operating as designed, and identify problems before the dreaded phone call. To enable this level of robust testing, pyATS offers a standard way to communicate with network elements and standardize the data returned into native Python objects. This core functionality opens up a lot of flexibility on how pyATS can be used by network developers.

In the following exercises, you will get a brief introduction to pyATS to connect and learn about device details.

### Connect and Interact with a Device

1. From the root of the python\_networking repository, change into the exercise directory.

* cd network\_testing/pyats

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

1. Import in pyATS libraries and tools

* from genie.conf import Genie  
  from ats.topology import loader  
  from genie.abstract import Lookup  
  from genie.libs import ops # noqa

1. Read and process the testbed (inventory) file

* genie\_testbed = Genie.init("./default\_testbed.yaml")

1. Create a pyATS device object from testbed

* vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()

1. Execute a NETCONF using this filter

* r = m.get\_config("running", filter)

1. Pretty print the raw XML to screen (expected output will not include the loopback interface, as you just deleted it)

* xml\_doc = minidom.parseString(r.xml)  
  print(xml\_doc.toprettyxml(indent = " "))

### End the NETCONF Connection

1. Send a RPC request to disconnect the connection.

* m.close\_session()  
  m.connected

1. End the Python interpreter.

* exit()

## CLI with netmiko

1. From the root of the python\_networking repository, change into the exercise directory.

* cd device\_apis/cli

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

### Retrieve Network Configuration Details with CLI with netmiko\_example1.py

1. Import libraries

* from netmiko import ConnectHandler  
  import re  
  import sys

1. Add parent directory to path to allow importing common vars

* sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device

1. Set device\_type for netmiko

* device["device\_type"] = "cisco\_ios"

1. Create a CLI command template

* show\_interface\_config\_temp = "show running-config interface {}"

1. Open CLI connection to device.
   * *Note: Normally you’d use a with block to open connection to device. This avoids needing to manually m.close\_session() at the end of a script, but for interactive use, this format is chosen.*

* ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create desired CLI command

* command = show\_interface\_config\_temp.format("GigabitEthernet2")

1. Verify the command has been created correctly

* print(command)

1. Send command to device

* interface = ch.send\_command(command)

1. Print the raw command output to the screen

* print(interface)

1. Create regular expression searches to parse the output for desired interface details

* name = re.search(r'interface (.\*)', interface).group(1)  
  description = re.search(r'description (.\*)', interface).group(1)

1. Pull out the ip and mask for the interface

* ip\_info = re.search(r'ip address (.\*) (.\*)', interface)  
  ip = ip\_info.group(1)  
  netmask = ip\_info.group(2)

1. Print the desired info to the screen

* print("The interface {name} has ip address {ip}/{mask}".format(  
   name = name,  
   ip = ip,  
   mask = netmask,  
   )  
   )

### Modify Network Configuration Details with CLI with netmiko\_example2.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from netmiko import ConnectHandler  
  import re, sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  device["device\_type"] = "cisco\_ios"  
  show\_interface\_config\_temp = "show running-config interface {}"  
  ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create Python dictionary with new Loopback Details

* loopback = {"int\_name": "Loopback103",  
   "description": "Demo interface by CLI and netmiko",  
   "ip": "192.168.103.1",  
   "netmask": "255.255.255.0"}

1. Create a CLI configuration

* interface\_config = [  
   "interface {}".format(loopback["int\_name"]),  
   "description {}".format(loopback["description"]),  
   "ip address {} {}".format(loopback["ip"], loopback["netmask"]),  
   "no shut"  
  ]

1. Send configuration to device

* output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to retrieve the new configuration.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)

### Delete Network Configuration Details with CLI with netmiko\_example3.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from netmiko import ConnectHandler  
  import re, sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  device["device\_type"] = "cisco\_ios"  
  show\_interface\_config\_temp = "show running-config interface {}"  
  ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create a new CLI configuration to delete the interface.

* interface\_config = [  
   "no interface {}".format(loopback["int\_name"])  
  ]

1. Send configuration to device

* output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to verify configuration removed.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)
* *Note: attempting to view the configuration of a non-existing interface will generate a CLI error. This output is expected, and one of the reasons APIs like NETCONF or RESTCONF are better suited to programmatic interactions.*

### End the CLI connection to the device

1. Disconnect from the device.

* ch.disconnect()

1. End the Python interpreter.

* exit()

## Other Cool Python Stuff

### Introduction to pyATS

[pyATS](https://developer.cisco.com/site/pyats) is a network testing tool developed by Cisco and made available for free, with significant elements of the underlying code open source.

pyATS offers network developers the ability to profile the network state of hardware, interfaces, protocols, etc… before, during and after changes, to ensure the network is operating as designed, and identify problems before the dreaded phone call. To enable this level of robust testing, pyATS offers a standard way to communicate with network elements and standardize the data returned into native Python objects. This core functionality opens up a lot of flexibility on how pyATS can be used by network developers.

In the following exercises, you will get a brief introduction to pyATS to connect and learn about device details.

### Connect and Interact with a Device

1. From the root of the python\_networking repository, change into the exercise directory.

* cd network\_testing/pyats

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

1. Import in pyATS libraries and tools

* from genie.conf import Genie  
  from ats.topology import loader  
  from genie.abstract import Lookup  
  from genie.libs import ops # noqa

1. Read and process the testbed (inventory) file

* genie\_testbed = Genie.init("./default\_testbed.yaml")

1. Create a pyATS device object from testbed

* vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()
* filter = interface\_filter.format(int\_name = "Loopback102")

1. Execute a NETCONF using this filter

* r = m.get\_config("running", filter)

1. Pretty print the raw XML to screen (expected output will not include the loopback interface, as you just deleted it)

* xml\_doc = minidom.parseString(r.xml)  
  print(xml\_doc.toprettyxml(indent = " "))

### End the NETCONF Connection

1. Send a RPC request to disconnect the connection.

* m.close\_session()  
  m.connected

1. End the Python interpreter.

* exit()

## CLI with netmiko

1. From the root of the python\_networking repository, change into the exercise directory.

* cd device\_apis/cli

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

### Retrieve Network Configuration Details with CLI with netmiko\_example1.py

1. Import libraries

* from netmiko import ConnectHandler  
  import re  
  import sys

1. Add parent directory to path to allow importing common vars

* sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device

1. Set device\_type for netmiko

* device["device\_type"] = "cisco\_ios"

1. Create a CLI command template

* show\_interface\_config\_temp = "show running-config interface {}"

1. Open CLI connection to device.
   * *Note: Normally you’d use a with block to open connection to device. This avoids needing to manually m.close\_session() at the end of a script, but for interactive use, this format is chosen.*

* ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create desired CLI command

* command = show\_interface\_config\_temp.format("GigabitEthernet2")

1. Verify the command has been created correctly

* print(command)

1. Send command to device

* interface = ch.send\_command(command)

1. Print the raw command output to the screen

* print(interface)

1. Create regular expression searches to parse the output for desired interface details

* name = re.search(r'interface (.\*)', interface).group(1)  
  description = re.search(r'description (.\*)', interface).group(1)

1. Pull out the ip and mask for the interface

* ip\_info = re.search(r'ip address (.\*) (.\*)', interface)  
  ip = ip\_info.group(1)  
  netmask = ip\_info.group(2)

1. Print the desired info to the screen

* print("The interface {name} has ip address {ip}/{mask}".format(  
   name = name,  
   ip = ip,  
   mask = netmask,  
   )  
   )

### Modify Network Configuration Details with CLI with netmiko\_example2.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from netmiko import ConnectHandler  
  import re, sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  device["device\_type"] = "cisco\_ios"  
  show\_interface\_config\_temp = "show running-config interface {}"  
  ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create Python dictionary with new Loopback Details

* loopback = {"int\_name": "Loopback103",  
   "description": "Demo interface by CLI and netmiko",  
   "ip": "192.168.103.1",  
   "netmask": "255.255.255.0"}

1. Create a CLI configuration

* interface\_config = [  
   "interface {}".format(loopback["int\_name"]),  
   "description {}".format(loopback["description"]),  
   "ip address {} {}".format(loopback["ip"], loopback["netmask"]),  
   "no shut"  
  ]

1. Send configuration to device

* output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to retrieve the new configuration.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)

### Delete Network Configuration Details with CLI with netmiko\_example3.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from netmiko import ConnectHandler  
  import re, sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  device["device\_type"] = "cisco\_ios"  
  show\_interface\_config\_temp = "show running-config interface {}"  
  ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create a new CLI configuration to delete the interface.

* interface\_config = [  
   "no interface {}".format(loopback["int\_name"])  
  ]

1. Send configuration to device

* output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to verify configuration removed.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)
* *Note: attempting to view the configuration of a non-existing interface will generate a CLI error. This output is expected, and one of the reasons APIs like NETCONF or RESTCONF are better suited to programmatic interactions.*

### End the CLI connection to the device

1. Disconnect from the device.

* ch.disconnect()

1. End the Python interpreter.

* exit()

## Other Cool Python Stuff

### Introduction to pyATS

[pyATS](https://developer.cisco.com/site/pyats) is a network testing tool developed by Cisco and made available for free, with significant elements of the underlying code open source.

pyATS offers network developers the ability to profile the network state of hardware, interfaces, protocols, etc… before, during and after changes, to ensure the network is operating as designed, and identify problems before the dreaded phone call. To enable this level of robust testing, pyATS offers a standard way to communicate with network elements and standardize the data returned into native Python objects. This core functionality opens up a lot of flexibility on how pyATS can be used by network developers.

In the following exercises, you will get a brief introduction to pyATS to connect and learn about device details.

### Connect and Interact with a Device

1. From the root of the python\_networking repository, change into the exercise directory.

* cd network\_testing/pyats

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

1. Import in pyATS libraries and tools

* from genie.conf import Genie  
  from ats.topology import loader  
  from genie.abstract import Lookup  
  from genie.libs import ops # noqa

1. Read and process the testbed (inventory) file

* genie\_testbed = Genie.init("./default\_testbed.yaml")

1. Create a pyATS device object from testbed

* vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()
* xml\_doc = minidom.parseString(r.xml)  
  print(xml\_doc.toprettyxml(indent = " "))

### Delete Network Configuration Details with NETCONF with netconf\_example3.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from ncclient import manager  
  from xml.dom import minidom  
  import xmltodict  
  import sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  interface\_filter = """  
  <filter>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </filter>  
  """  
  loopback = {"int\_name": "Loopback102",  
   "description": "Demo interface by NETCONF",  
   "ip": "192.168.102.1",  
   "netmask": "255.255.255.0"}  
  m = manager.connect(host = device["address"],  
   port = device["netconf\_port"],  
   username = device["username"],  
   password = device["password"],  
   hostkey\_verify = False)

1. Verify NETCONF connection is active

* m.connected

1. Create new config template to delete an interface

* config\_data = """  
  <config>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface operation="delete">  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </config>  
  """

1. Create desired NETCONF config payload and execute to delete the interface

* config = config\_data.format(\*\*loopback)  
  r = m.edit\_config(target = "running", config = config)

1. Print OK status (expected output true)

* print("NETCONF RPC OK: {}".format(r.ok))

1. Create a new NETCONF to check on new loopback interface

* filter = interface\_filter.format(int\_name = "Loopback102")

1. Execute a NETCONF using this filter

* r = m.get\_config("running", filter)

1. Pretty print the raw XML to screen (expected output will not include the loopback interface, as you just deleted it)

* xml\_doc = minidom.parseString(r.xml)  
  print(xml\_doc.toprettyxml(indent = " "))

### End the NETCONF Connection

1. Send a RPC request to disconnect the connection.

* m.close\_session()  
  m.connected

1. End the Python interpreter.

* exit()

## CLI with netmiko

1. From the root of the python\_networking repository, change into the exercise directory.

* cd device\_apis/cli

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

### Retrieve Network Configuration Details with CLI with netmiko\_example1.py

1. Import libraries

* from netmiko import ConnectHandler  
  import re  
  import sys

1. Add parent directory to path to allow importing common vars

* sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device

1. Set device\_type for netmiko

* device["device\_type"] = "cisco\_ios"

1. Create a CLI command template

* show\_interface\_config\_temp = "show running-config interface {}"

1. Open CLI connection to device.
   * *Note: Normally you’d use a with block to open connection to device. This avoids needing to manually m.close\_session() at the end of a script, but for interactive use, this format is chosen.*

* ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create desired CLI command

* command = show\_interface\_config\_temp.format("GigabitEthernet2")

1. Verify the command has been created correctly

* print(command)

1. Send command to device

* interface = ch.send\_command(command)

1. Print the raw command output to the screen

* print(interface)

1. Create regular expression searches to parse the output for desired interface details

* name = re.search(r'interface (.\*)', interface).group(1)  
  description = re.search(r'description (.\*)', interface).group(1)

1. Pull out the ip and mask for the interface

* ip\_info = re.search(r'ip address (.\*) (.\*)', interface)  
  ip = ip\_info.group(1)  
  netmask = ip\_info.group(2)

1. Print the desired info to the screen

* print("The interface {name} has ip address {ip}/{mask}".format(  
   name = name,  
   ip = ip,  
   mask = netmask,  
   )  
   )

### Modify Network Configuration Details with CLI with netmiko\_example2.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from netmiko import ConnectHandler  
  import re, sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  device["device\_type"] = "cisco\_ios"  
  show\_interface\_config\_temp = "show running-config interface {}"  
  ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create Python dictionary with new Loopback Details

* loopback = {"int\_name": "Loopback103",  
   "description": "Demo interface by CLI and netmiko",  
   "ip": "192.168.103.1",  
   "netmask": "255.255.255.0"}

1. Create a CLI configuration

* interface\_config = [  
   "interface {}".format(loopback["int\_name"]),  
   "description {}".format(loopback["description"]),  
   "ip address {} {}".format(loopback["ip"], loopback["netmask"]),  
   "no shut"  
  ]

1. Send configuration to device

* output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to retrieve the new configuration.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)

### Delete Network Configuration Details with CLI with netmiko\_example3.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from netmiko import ConnectHandler  
  import re, sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  device["device\_type"] = "cisco\_ios"  
  show\_interface\_config\_temp = "show running-config interface {}"  
  ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create a new CLI configuration to delete the interface.

* interface\_config = [  
   "no interface {}".format(loopback["int\_name"])  
  ]

1. Send configuration to device

* output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to verify configuration removed.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)
* *Note: attempting to view the configuration of a non-existing interface will generate a CLI error. This output is expected, and one of the reasons APIs like NETCONF or RESTCONF are better suited to programmatic interactions.*

### End the CLI connection to the device

1. Disconnect from the device.

* ch.disconnect()

1. End the Python interpreter.

* exit()

## Other Cool Python Stuff

### Introduction to pyATS

[pyATS](https://developer.cisco.com/site/pyats) is a network testing tool developed by Cisco and made available for free, with significant elements of the underlying code open source.

pyATS offers network developers the ability to profile the network state of hardware, interfaces, protocols, etc… before, during and after changes, to ensure the network is operating as designed, and identify problems before the dreaded phone call. To enable this level of robust testing, pyATS offers a standard way to communicate with network elements and standardize the data returned into native Python objects. This core functionality opens up a lot of flexibility on how pyATS can be used by network developers.

In the following exercises, you will get a brief introduction to pyATS to connect and learn about device details.

### Connect and Interact with a Device

1. From the root of the python\_networking repository, change into the exercise directory.

* cd network\_testing/pyats

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

1. Import in pyATS libraries and tools

* from genie.conf import Genie  
  from ats.topology import loader  
  from genie.abstract import Lookup  
  from genie.libs import ops # noqa

1. Read and process the testbed (inventory) file

* genie\_testbed = Genie.init("./default\_testbed.yaml")

1. Create a pyATS device object from testbed

* vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()

## NETCONF with ncclient

1. From the root of the python\_networking repository, change into the exercise directory.

* cd device\_apis/netconf

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

### Retrieve Network Configuration Details with NETCONF with netconf\_example1.py

1. Import libraries

* from ncclient import manager  
  from xml.dom import minidom  
  import xmltodict  
  import sys

1. Add parent directory to path to allow importing common vars

* sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device

1. Create filter template for an interface

* interface\_filter = """  
  <filter>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </filter>  
  """

1. Open NETCONF connection to device
   * *Note: Normally you’d use a with block to open connection to device. This avoids needing to manually m.close\_session() at the end of a script, but for interactive use, this format is chosen.*

* m = manager.connect(host = device["address"],  
   port = device["netconf\_port"],  
   username = device["username"],  
   password = device["password"],  
   hostkey\_verify = False)

1. Verify NETCONF connection is active (expected output true)

* m.connected

1. Create desired NETCONF filter for a particular interface

* filter = interface\_filter.format(int\_name = "GigabitEthernet2")

1. Execute a NETCONF using the filter

* r = m.get\_config("running", filter)

1. Pretty print raw xml to screen

* xml\_doc = minidom.parseString(r.xml)  
  print(xml\_doc.toprettyxml(indent = " "))

1. Process the XML data into Python Dictionary and use

* interface = xmltodict.parse(r.xml)

1. Pretty Print the full Python (Ordered) Dictionary.

* from pprint import pprint  
  pprint(interface)

1. If RPC returned data, print out the interesting pieces.

* if not interface["rpc-reply"]["data"] is None:  
   # Create Python variable for interface details  
   interface = interface["rpc-reply"]["data"]["interfaces"]["interface"]  
    
   print("The interface {name} has ip address {ip}/{mask}".format(  
   name = interface["name"]["#text"],  
   ip = interface["ipv4"]["address"]["ip"],  
   mask = interface["ipv4"]["address"]["netmask"],  
   )  
   )  
  else:  
   print("No interface {} found".format("GigabitEthernet2"))

### Modify Network Configuration Details with NETCONF with netconf\_example2.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from ncclient import manager  
  from xml.dom import minidom  
  import xmltodict  
  import sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  interface\_filter = """  
  <filter>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </filter>  
  """  
  m = manager.connect(host = device["address"],  
   port = device["netconf\_port"],  
   username = device["username"],  
   password = device["password"],  
   hostkey\_verify = False)

1. Verify NETCONF connection is active

* m.connected

1. Create Python dictionary with new Loopback Details

* loopback = {"int\_name": "Loopback102",  
   "description": "Demo interface by NETCONF",  
   "ip": "192.168.102.1",  
   "netmask": "255.255.255.0"}

1. Create NETCONF template for an interface

* config\_data = """  
  <config>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   <description>{description}</description>  
   <type xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">  
   ianaift:softwareLoopback  
   </type>  
   <enabled>true</enabled>  
   <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">  
   <address>  
   <ip>{ip}</ip>  
   <netmask>{netmask}</netmask>  
   </address>  
   </ipv4>  
   </interface>  
   </interfaces>  
  </config>  
  """

1. Create desired NETCONF config payload

* config = config\_data.format(\*\*loopback)

1. Send operation

* r = m.edit\_config(target = "running", config = config)

1. Print OK status (expected output true)

* print("NETCONF RPC OK: {}".format(r.ok))

1. Create a new NETCONF to check on new loopback interface

* filter = interface\_filter.format(int\_name = "Loopback102")

1. Execute a NETCONF using this filter

* r = m.get\_config("running", filter)

1. Pretty print the raw XML to screen

* xml\_doc = minidom.parseString(r.xml)  
  print(xml\_doc.toprettyxml(indent = " "))

### Delete Network Configuration Details with NETCONF with netconf\_example3.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from ncclient import manager  
  from xml.dom import minidom  
  import xmltodict  
  import sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  interface\_filter = """  
  <filter>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </filter>  
  """  
  loopback = {"int\_name": "Loopback102",  
   "description": "Demo interface by NETCONF",  
   "ip": "192.168.102.1",  
   "netmask": "255.255.255.0"}  
  m = manager.connect(host = device["address"],  
   port = device["netconf\_port"],  
   username = device["username"],  
   password = device["password"],  
   hostkey\_verify = False)

1. Verify NETCONF connection is active

* m.connected

1. Create new config template to delete an interface

* config\_data = """  
  <config>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface operation="delete">  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </config>  
  """

1. Create desired NETCONF config payload and execute to delete the interface

* config = config\_data.format(\*\*loopback)  
  r = m.edit\_config(target = "running", config = config)

1. Print OK status (expected output true)

* print("NETCONF RPC OK: {}".format(r.ok))

1. Create a new NETCONF to check on new loopback interface

* filter = interface\_filter.format(int\_name = "Loopback102")

1. Execute a NETCONF using this filter

* r = m.get\_config("running", filter)

1. Pretty print the raw XML to screen (expected output will not include the loopback interface, as you just deleted it)

* xml\_doc = minidom.parseString(r.xml)  
  print(xml\_doc.toprettyxml(indent = " "))

### End the NETCONF Connection

1. Send a RPC request to disconnect the connection.

* m.close\_session()  
  m.connected

1. End the Python interpreter.

* exit()

## CLI with netmiko

1. From the root of the python\_networking repository, change into the exercise directory.

* cd device\_apis/cli

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

### Retrieve Network Configuration Details with CLI with netmiko\_example1.py

1. Import libraries

* from netmiko import ConnectHandler  
  import re  
  import sys

1. Add parent directory to path to allow importing common vars

* sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device

1. Set device\_type for netmiko

* device["device\_type"] = "cisco\_ios"

1. Create a CLI command template

* show\_interface\_config\_temp = "show running-config interface {}"

1. Open CLI connection to device.
   * *Note: Normally you’d use a with block to open connection to device. This avoids needing to manually m.close\_session() at the end of a script, but for interactive use, this format is chosen.*

* ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create desired CLI command

* command = show\_interface\_config\_temp.format("GigabitEthernet2")

1. Verify the command has been created correctly

* print(command)

1. Send command to device

* interface = ch.send\_command(command)

1. Print the raw command output to the screen

* print(interface)

1. Create regular expression searches to parse the output for desired interface details

* name = re.search(r'interface (.\*)', interface).group(1)  
  description = re.search(r'description (.\*)', interface).group(1)

1. Pull out the ip and mask for the interface

* ip\_info = re.search(r'ip address (.\*) (.\*)', interface)  
  ip = ip\_info.group(1)  
  netmask = ip\_info.group(2)

1. Print the desired info to the screen

* print("The interface {name} has ip address {ip}/{mask}".format(  
   name = name,  
   ip = ip,  
   mask = netmask,  
   )  
   )

### Modify Network Configuration Details with CLI with netmiko\_example2.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from netmiko import ConnectHandler  
  import re, sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  device["device\_type"] = "cisco\_ios"  
  show\_interface\_config\_temp = "show running-config interface {}"  
  ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create Python dictionary with new Loopback Details

* loopback = {"int\_name": "Loopback103",  
   "description": "Demo interface by CLI and netmiko",  
   "ip": "192.168.103.1",  
   "netmask": "255.255.255.0"}

1. Create a CLI configuration

* interface\_config = [  
   "interface {}".format(loopback["int\_name"]),  
   "description {}".format(loopback["description"]),  
   "ip address {} {}".format(loopback["ip"], loopback["netmask"]),  
   "no shut"  
  ]

1. Send configuration to device

* output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to retrieve the new configuration.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)

### Delete Network Configuration Details with CLI with netmiko\_example3.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from netmiko import ConnectHandler  
  import re, sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  device["device\_type"] = "cisco\_ios"  
  show\_interface\_config\_temp = "show running-config interface {}"  
  ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create a new CLI configuration to delete the interface.

* interface\_config = [  
   "no interface {}".format(loopback["int\_name"])  
  ]

1. Send configuration to device

* output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to verify configuration removed.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)
* *Note: attempting to view the configuration of a non-existing interface will generate a CLI error. This output is expected, and one of the reasons APIs like NETCONF or RESTCONF are better suited to programmatic interactions.*

### End the CLI connection to the device

1. Disconnect from the device.

* ch.disconnect()

1. End the Python interpreter.

* exit()

## Other Cool Python Stuff

### Introduction to pyATS

[pyATS](https://developer.cisco.com/site/pyats) is a network testing tool developed by Cisco and made available for free, with significant elements of the underlying code open source.

pyATS offers network developers the ability to profile the network state of hardware, interfaces, protocols, etc… before, during and after changes, to ensure the network is operating as designed, and identify problems before the dreaded phone call. To enable this level of robust testing, pyATS offers a standard way to communicate with network elements and standardize the data returned into native Python objects. This core functionality opens up a lot of flexibility on how pyATS can be used by network developers.

In the following exercises, you will get a brief introduction to pyATS to connect and learn about device details.

### Connect and Interact with a Device

1. From the root of the python\_networking repository, change into the exercise directory.

* cd network\_testing/pyats

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

1. Import in pyATS libraries and tools

* from genie.conf import Genie  
  from ats.topology import loader  
  from genie.abstract import Lookup  
  from genie.libs import ops # noqa

1. Read and process the testbed (inventory) file

* genie\_testbed = Genie.init("./default\_testbed.yaml")

1. Create a pyATS device object from testbed

* vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()
* print(r.status\_code)

## NETCONF with ncclient

1. From the root of the python\_networking repository, change into the exercise directory.

* cd device\_apis/netconf

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

### Retrieve Network Configuration Details with NETCONF with netconf\_example1.py

1. Import libraries

* from ncclient import manager  
  from xml.dom import minidom  
  import xmltodict  
  import sys

1. Add parent directory to path to allow importing common vars

* sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device

1. Create filter template for an interface

* interface\_filter = """  
  <filter>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </filter>  
  """

1. Open NETCONF connection to device
   * *Note: Normally you’d use a with block to open connection to device. This avoids needing to manually m.close\_session() at the end of a script, but for interactive use, this format is chosen.*

* m = manager.connect(host = device["address"],  
   port = device["netconf\_port"],  
   username = device["username"],  
   password = device["password"],  
   hostkey\_verify = False)

1. Verify NETCONF connection is active (expected output true)

* m.connected

1. Create desired NETCONF filter for a particular interface

* filter = interface\_filter.format(int\_name = "GigabitEthernet2")

1. Execute a NETCONF using the filter

* r = m.get\_config("running", filter)

1. Pretty print raw xml to screen

* xml\_doc = minidom.parseString(r.xml)  
  print(xml\_doc.toprettyxml(indent = " "))

1. Process the XML data into Python Dictionary and use

* interface = xmltodict.parse(r.xml)

1. Pretty Print the full Python (Ordered) Dictionary.

* from pprint import pprint  
  pprint(interface)

1. If RPC returned data, print out the interesting pieces.

* if not interface["rpc-reply"]["data"] is None:  
   # Create Python variable for interface details  
   interface = interface["rpc-reply"]["data"]["interfaces"]["interface"]  
    
   print("The interface {name} has ip address {ip}/{mask}".format(  
   name = interface["name"]["#text"],  
   ip = interface["ipv4"]["address"]["ip"],  
   mask = interface["ipv4"]["address"]["netmask"],  
   )  
   )  
  else:  
   print("No interface {} found".format("GigabitEthernet2"))

### Modify Network Configuration Details with NETCONF with netconf\_example2.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from ncclient import manager  
  from xml.dom import minidom  
  import xmltodict  
  import sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  interface\_filter = """  
  <filter>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </filter>  
  """  
  m = manager.connect(host = device["address"],  
   port = device["netconf\_port"],  
   username = device["username"],  
   password = device["password"],  
   hostkey\_verify = False)

1. Verify NETCONF connection is active

* m.connected

1. Create Python dictionary with new Loopback Details

* loopback = {"int\_name": "Loopback102",  
   "description": "Demo interface by NETCONF",  
   "ip": "192.168.102.1",  
   "netmask": "255.255.255.0"}

1. Create NETCONF template for an interface

* config\_data = """  
  <config>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   <description>{description}</description>  
   <type xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">  
   ianaift:softwareLoopback  
   </type>  
   <enabled>true</enabled>  
   <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">  
   <address>  
   <ip>{ip}</ip>  
   <netmask>{netmask}</netmask>  
   </address>  
   </ipv4>  
   </interface>  
   </interfaces>  
  </config>  
  """

1. Create desired NETCONF config payload

* config = config\_data.format(\*\*loopback)

1. Send operation

* r = m.edit\_config(target = "running", config = config)

1. Print OK status (expected output true)

* print("NETCONF RPC OK: {}".format(r.ok))

1. Create a new NETCONF to check on new loopback interface

* filter = interface\_filter.format(int\_name = "Loopback102")

1. Execute a NETCONF using this filter

* r = m.get\_config("running", filter)

1. Pretty print the raw XML to screen

* xml\_doc = minidom.parseString(r.xml)  
  print(xml\_doc.toprettyxml(indent = " "))

### Delete Network Configuration Details with NETCONF with netconf\_example3.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from ncclient import manager  
  from xml.dom import minidom  
  import xmltodict  
  import sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  interface\_filter = """  
  <filter>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </filter>  
  """  
  loopback = {"int\_name": "Loopback102",  
   "description": "Demo interface by NETCONF",  
   "ip": "192.168.102.1",  
   "netmask": "255.255.255.0"}  
  m = manager.connect(host = device["address"],  
   port = device["netconf\_port"],  
   username = device["username"],  
   password = device["password"],  
   hostkey\_verify = False)

1. Verify NETCONF connection is active

* m.connected

1. Create new config template to delete an interface

* config\_data = """  
  <config>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface operation="delete">  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </config>  
  """

1. Create desired NETCONF config payload and execute to delete the interface

* config = config\_data.format(\*\*loopback)  
  r = m.edit\_config(target = "running", config = config)

1. Print OK status (expected output true)

* print("NETCONF RPC OK: {}".format(r.ok))

1. Create a new NETCONF to check on new loopback interface

* filter = interface\_filter.format(int\_name = "Loopback102")

1. Execute a NETCONF using this filter

* r = m.get\_config("running", filter)

1. Pretty print the raw XML to screen (expected output will not include the loopback interface, as you just deleted it)

* xml\_doc = minidom.parseString(r.xml)  
  print(xml\_doc.toprettyxml(indent = " "))

### End the NETCONF Connection

1. Send a RPC request to disconnect the connection.

* m.close\_session()  
  m.connected

1. End the Python interpreter.

* exit()

## CLI with netmiko

1. From the root of the python\_networking repository, change into the exercise directory.

* cd device\_apis/cli

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

### Retrieve Network Configuration Details with CLI with netmiko\_example1.py

1. Import libraries

* from netmiko import ConnectHandler  
  import re  
  import sys

1. Add parent directory to path to allow importing common vars

* sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device

1. Set device\_type for netmiko

* device["device\_type"] = "cisco\_ios"

1. Create a CLI command template

* show\_interface\_config\_temp = "show running-config interface {}"

1. Open CLI connection to device.
   * *Note: Normally you’d use a with block to open connection to device. This avoids needing to manually m.close\_session() at the end of a script, but for interactive use, this format is chosen.*

* ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create desired CLI command

* command = show\_interface\_config\_temp.format("GigabitEthernet2")

1. Verify the command has been created correctly

* print(command)

1. Send command to device

* interface = ch.send\_command(command)

1. Print the raw command output to the screen

* print(interface)

1. Create regular expression searches to parse the output for desired interface details

* name = re.search(r'interface (.\*)', interface).group(1)  
  description = re.search(r'description (.\*)', interface).group(1)

1. Pull out the ip and mask for the interface

* ip\_info = re.search(r'ip address (.\*) (.\*)', interface)  
  ip = ip\_info.group(1)  
  netmask = ip\_info.group(2)

1. Print the desired info to the screen

* print("The interface {name} has ip address {ip}/{mask}".format(  
   name = name,  
   ip = ip,  
   mask = netmask,  
   )  
   )

### Modify Network Configuration Details with CLI with netmiko\_example2.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from netmiko import ConnectHandler  
  import re, sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  device["device\_type"] = "cisco\_ios"  
  show\_interface\_config\_temp = "show running-config interface {}"  
  ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create Python dictionary with new Loopback Details

* loopback = {"int\_name": "Loopback103",  
   "description": "Demo interface by CLI and netmiko",  
   "ip": "192.168.103.1",  
   "netmask": "255.255.255.0"}

1. Create a CLI configuration

* interface\_config = [  
   "interface {}".format(loopback["int\_name"]),  
   "description {}".format(loopback["description"]),  
   "ip address {} {}".format(loopback["ip"], loopback["netmask"]),  
   "no shut"  
  ]

1. Send configuration to device

* output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to retrieve the new configuration.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)

### Delete Network Configuration Details with CLI with netmiko\_example3.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from netmiko import ConnectHandler  
  import re, sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  device["device\_type"] = "cisco\_ios"  
  show\_interface\_config\_temp = "show running-config interface {}"  
  ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create a new CLI configuration to delete the interface.

* interface\_config = [  
   "no interface {}".format(loopback["int\_name"])  
  ]

1. Send configuration to device

* output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to verify configuration removed.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)
* *Note: attempting to view the configuration of a non-existing interface will generate a CLI error. This output is expected, and one of the reasons APIs like NETCONF or RESTCONF are better suited to programmatic interactions.*

### End the CLI connection to the device

1. Disconnect from the device.

* ch.disconnect()

1. End the Python interpreter.

* exit()

## Other Cool Python Stuff

### Introduction to pyATS

[pyATS](https://developer.cisco.com/site/pyats) is a network testing tool developed by Cisco and made available for free, with significant elements of the underlying code open source.

pyATS offers network developers the ability to profile the network state of hardware, interfaces, protocols, etc… before, during and after changes, to ensure the network is operating as designed, and identify problems before the dreaded phone call. To enable this level of robust testing, pyATS offers a standard way to communicate with network elements and standardize the data returned into native Python objects. This core functionality opens up a lot of flexibility on how pyATS can be used by network developers.

In the following exercises, you will get a brief introduction to pyATS to connect and learn about device details.

### Connect and Interact with a Device

1. From the root of the python\_networking repository, change into the exercise directory.

* cd network\_testing/pyats

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

1. Import in pyATS libraries and tools

* from genie.conf import Genie  
  from ats.topology import loader  
  from genie.abstract import Lookup  
  from genie.libs import ops # noqa

1. Read and process the testbed (inventory) file

* genie\_testbed = Genie.init("./default\_testbed.yaml")

1. Create a pyATS device object from testbed

* vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()

1. Check status code (expected 404)

* print(r.status\_code)

## NETCONF with ncclient

1. From the root of the python\_networking repository, change into the exercise directory.

* cd device\_apis/netconf

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

### Retrieve Network Configuration Details with NETCONF with netconf\_example1.py

1. Import libraries

* from ncclient import manager  
  from xml.dom import minidom  
  import xmltodict  
  import sys

1. Add parent directory to path to allow importing common vars

* sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device

1. Create filter template for an interface

* interface\_filter = """  
  <filter>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </filter>  
  """

1. Open NETCONF connection to device
   * *Note: Normally you’d use a with block to open connection to device. This avoids needing to manually m.close\_session() at the end of a script, but for interactive use, this format is chosen.*

* m = manager.connect(host = device["address"],  
   port = device["netconf\_port"],  
   username = device["username"],  
   password = device["password"],  
   hostkey\_verify = False)

1. Verify NETCONF connection is active (expected output true)

* m.connected

1. Create desired NETCONF filter for a particular interface

* filter = interface\_filter.format(int\_name = "GigabitEthernet2")

1. Execute a NETCONF using the filter

* r = m.get\_config("running", filter)

1. Pretty print raw xml to screen

* xml\_doc = minidom.parseString(r.xml)  
  print(xml\_doc.toprettyxml(indent = " "))

1. Process the XML data into Python Dictionary and use

* interface = xmltodict.parse(r.xml)

1. Pretty Print the full Python (Ordered) Dictionary.

* from pprint import pprint  
  pprint(interface)

1. If RPC returned data, print out the interesting pieces.

* if not interface["rpc-reply"]["data"] is None:  
   # Create Python variable for interface details  
   interface = interface["rpc-reply"]["data"]["interfaces"]["interface"]  
    
   print("The interface {name} has ip address {ip}/{mask}".format(  
   name = interface["name"]["#text"],  
   ip = interface["ipv4"]["address"]["ip"],  
   mask = interface["ipv4"]["address"]["netmask"],  
   )  
   )  
  else:  
   print("No interface {} found".format("GigabitEthernet2"))

### Modify Network Configuration Details with NETCONF with netconf\_example2.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from ncclient import manager  
  from xml.dom import minidom  
  import xmltodict  
  import sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  interface\_filter = """  
  <filter>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </filter>  
  """  
  m = manager.connect(host = device["address"],  
   port = device["netconf\_port"],  
   username = device["username"],  
   password = device["password"],  
   hostkey\_verify = False)

1. Verify NETCONF connection is active

* m.connected

1. Create Python dictionary with new Loopback Details

* loopback = {"int\_name": "Loopback102",  
   "description": "Demo interface by NETCONF",  
   "ip": "192.168.102.1",  
   "netmask": "255.255.255.0"}

1. Create NETCONF template for an interface

* config\_data = """  
  <config>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   <description>{description}</description>  
   <type xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">  
   ianaift:softwareLoopback  
   </type>  
   <enabled>true</enabled>  
   <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">  
   <address>  
   <ip>{ip}</ip>  
   <netmask>{netmask}</netmask>  
   </address>  
   </ipv4>  
   </interface>  
   </interfaces>  
  </config>  
  """

1. Create desired NETCONF config payload

* config = config\_data.format(\*\*loopback)

1. Send operation

* r = m.edit\_config(target = "running", config = config)

1. Print OK status (expected output true)

* print("NETCONF RPC OK: {}".format(r.ok))

1. Create a new NETCONF to check on new loopback interface

* filter = interface\_filter.format(int\_name = "Loopback102")

1. Execute a NETCONF using this filter

* r = m.get\_config("running", filter)

1. Pretty print the raw XML to screen

* xml\_doc = minidom.parseString(r.xml)  
  print(xml\_doc.toprettyxml(indent = " "))

### Delete Network Configuration Details with NETCONF with netconf\_example3.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from ncclient import manager  
  from xml.dom import minidom  
  import xmltodict  
  import sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  interface\_filter = """  
  <filter>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </filter>  
  """  
  loopback = {"int\_name": "Loopback102",  
   "description": "Demo interface by NETCONF",  
   "ip": "192.168.102.1",  
   "netmask": "255.255.255.0"}  
  m = manager.connect(host = device["address"],  
   port = device["netconf\_port"],  
   username = device["username"],  
   password = device["password"],  
   hostkey\_verify = False)

1. Verify NETCONF connection is active

* m.connected

1. Create new config template to delete an interface

* config\_data = """  
  <config>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface operation="delete">  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </config>  
  """

1. Create desired NETCONF config payload and execute to delete the interface

* config = config\_data.format(\*\*loopback)  
  r = m.edit\_config(target = "running", config = config)

1. Print OK status (expected output true)

* print("NETCONF RPC OK: {}".format(r.ok))

1. Create a new NETCONF to check on new loopback interface

* filter = interface\_filter.format(int\_name = "Loopback102")

1. Execute a NETCONF using this filter

* r = m.get\_config("running", filter)

1. Pretty print the raw XML to screen (expected output will not include the loopback interface, as you just deleted it)

* xml\_doc = minidom.parseString(r.xml)  
  print(xml\_doc.toprettyxml(indent = " "))

### End the NETCONF Connection

1. Send a RPC request to disconnect the connection.

* m.close\_session()  
  m.connected

1. End the Python interpreter.

* exit()

## CLI with netmiko

1. From the root of the python\_networking repository, change into the exercise directory.

* cd device\_apis/cli

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

### Retrieve Network Configuration Details with CLI with netmiko\_example1.py

1. Import libraries

* from netmiko import ConnectHandler  
  import re  
  import sys

1. Add parent directory to path to allow importing common vars

* sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device

1. Set device\_type for netmiko

* device["device\_type"] = "cisco\_ios"

1. Create a CLI command template

* show\_interface\_config\_temp = "show running-config interface {}"

1. Open CLI connection to device.
   * *Note: Normally you’d use a with block to open connection to device. This avoids needing to manually m.close\_session() at the end of a script, but for interactive use, this format is chosen.*

* ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create desired CLI command

* command = show\_interface\_config\_temp.format("GigabitEthernet2")

1. Verify the command has been created correctly

* print(command)

1. Send command to device

* interface = ch.send\_command(command)

1. Print the raw command output to the screen

* print(interface)

1. Create regular expression searches to parse the output for desired interface details

* name = re.search(r'interface (.\*)', interface).group(1)  
  description = re.search(r'description (.\*)', interface).group(1)

1. Pull out the ip and mask for the interface

* ip\_info = re.search(r'ip address (.\*) (.\*)', interface)  
  ip = ip\_info.group(1)  
  netmask = ip\_info.group(2)

1. Print the desired info to the screen

* print("The interface {name} has ip address {ip}/{mask}".format(  
   name = name,  
   ip = ip,  
   mask = netmask,  
   )  
   )

### Modify Network Configuration Details with CLI with netmiko\_example2.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from netmiko import ConnectHandler  
  import re, sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  device["device\_type"] = "cisco\_ios"  
  show\_interface\_config\_temp = "show running-config interface {}"  
  ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create Python dictionary with new Loopback Details

* loopback = {"int\_name": "Loopback103",  
   "description": "Demo interface by CLI and netmiko",  
   "ip": "192.168.103.1",  
   "netmask": "255.255.255.0"}

1. Create a CLI configuration

* interface\_config = [  
   "interface {}".format(loopback["int\_name"]),  
   "description {}".format(loopback["description"]),  
   "ip address {} {}".format(loopback["ip"], loopback["netmask"]),  
   "no shut"  
  ]

1. Send configuration to device

* output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to retrieve the new configuration.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)

### Delete Network Configuration Details with CLI with netmiko\_example3.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from netmiko import ConnectHandler  
  import re, sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  device["device\_type"] = "cisco\_ios"  
  show\_interface\_config\_temp = "show running-config interface {}"  
  ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create a new CLI configuration to delete the interface.

* interface\_config = [  
   "no interface {}".format(loopback["int\_name"])  
  ]

1. Send configuration to device

* output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to verify configuration removed.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)
* *Note: attempting to view the configuration of a non-existing interface will generate a CLI error. This output is expected, and one of the reasons APIs like NETCONF or RESTCONF are better suited to programmatic interactions.*

### End the CLI connection to the device

1. Disconnect from the device.

* ch.disconnect()

1. End the Python interpreter.

* exit()

## Other Cool Python Stuff

### Introduction to pyATS

[pyATS](https://developer.cisco.com/site/pyats) is a network testing tool developed by Cisco and made available for free, with significant elements of the underlying code open source.

pyATS offers network developers the ability to profile the network state of hardware, interfaces, protocols, etc… before, during and after changes, to ensure the network is operating as designed, and identify problems before the dreaded phone call. To enable this level of robust testing, pyATS offers a standard way to communicate with network elements and standardize the data returned into native Python objects. This core functionality opens up a lot of flexibility on how pyATS can be used by network developers.

In the following exercises, you will get a brief introduction to pyATS to connect and learn about device details.

### Connect and Interact with a Device

1. From the root of the python\_networking repository, change into the exercise directory.

* cd network\_testing/pyats

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

1. Import in pyATS libraries and tools

* from genie.conf import Genie  
  from ats.topology import loader  
  from genie.abstract import Lookup  
  from genie.libs import ops # noqa

1. Read and process the testbed (inventory) file

* genie\_testbed = Genie.init("./default\_testbed.yaml")

1. Create a pyATS device object from testbed

* vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()
* r = requests.get(url,  
   headers = restconf\_headers,  
   auth=(device["username"], device["password"]),  
   verify=False)

1. Check status code (expected 404)

* print(r.status\_code)

## NETCONF with ncclient

1. From the root of the python\_networking repository, change into the exercise directory.

* cd device\_apis/netconf

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

### Retrieve Network Configuration Details with NETCONF with netconf\_example1.py

1. Import libraries

* from ncclient import manager  
  from xml.dom import minidom  
  import xmltodict  
  import sys

1. Add parent directory to path to allow importing common vars

* sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device

1. Create filter template for an interface

* interface\_filter = """  
  <filter>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </filter>  
  """

1. Open NETCONF connection to device
   * *Note: Normally you’d use a with block to open connection to device. This avoids needing to manually m.close\_session() at the end of a script, but for interactive use, this format is chosen.*

* m = manager.connect(host = device["address"],  
   port = device["netconf\_port"],  
   username = device["username"],  
   password = device["password"],  
   hostkey\_verify = False)

1. Verify NETCONF connection is active (expected output true)

* m.connected

1. Create desired NETCONF filter for a particular interface

* filter = interface\_filter.format(int\_name = "GigabitEthernet2")

1. Execute a NETCONF using the filter

* r = m.get\_config("running", filter)

1. Pretty print raw xml to screen

* xml\_doc = minidom.parseString(r.xml)  
  print(xml\_doc.toprettyxml(indent = " "))

1. Process the XML data into Python Dictionary and use

* interface = xmltodict.parse(r.xml)

1. Pretty Print the full Python (Ordered) Dictionary.

* from pprint import pprint  
  pprint(interface)

1. If RPC returned data, print out the interesting pieces.

* if not interface["rpc-reply"]["data"] is None:  
   # Create Python variable for interface details  
   interface = interface["rpc-reply"]["data"]["interfaces"]["interface"]  
    
   print("The interface {name} has ip address {ip}/{mask}".format(  
   name = interface["name"]["#text"],  
   ip = interface["ipv4"]["address"]["ip"],  
   mask = interface["ipv4"]["address"]["netmask"],  
   )  
   )  
  else:  
   print("No interface {} found".format("GigabitEthernet2"))

### Modify Network Configuration Details with NETCONF with netconf\_example2.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from ncclient import manager  
  from xml.dom import minidom  
  import xmltodict  
  import sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  interface\_filter = """  
  <filter>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </filter>  
  """  
  m = manager.connect(host = device["address"],  
   port = device["netconf\_port"],  
   username = device["username"],  
   password = device["password"],  
   hostkey\_verify = False)

1. Verify NETCONF connection is active

* m.connected

1. Create Python dictionary with new Loopback Details

* loopback = {"int\_name": "Loopback102",  
   "description": "Demo interface by NETCONF",  
   "ip": "192.168.102.1",  
   "netmask": "255.255.255.0"}

1. Create NETCONF template for an interface

* config\_data = """  
  <config>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   <description>{description}</description>  
   <type xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">  
   ianaift:softwareLoopback  
   </type>  
   <enabled>true</enabled>  
   <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">  
   <address>  
   <ip>{ip}</ip>  
   <netmask>{netmask}</netmask>  
   </address>  
   </ipv4>  
   </interface>  
   </interfaces>  
  </config>  
  """

1. Create desired NETCONF config payload

* config = config\_data.format(\*\*loopback)

1. Send operation

* r = m.edit\_config(target = "running", config = config)

1. Print OK status (expected output true)

* print("NETCONF RPC OK: {}".format(r.ok))

1. Create a new NETCONF to check on new loopback interface

* filter = interface\_filter.format(int\_name = "Loopback102")

1. Execute a NETCONF using this filter

* r = m.get\_config("running", filter)

1. Pretty print the raw XML to screen

* xml\_doc = minidom.parseString(r.xml)  
  print(xml\_doc.toprettyxml(indent = " "))

### Delete Network Configuration Details with NETCONF with netconf\_example3.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from ncclient import manager  
  from xml.dom import minidom  
  import xmltodict  
  import sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  interface\_filter = """  
  <filter>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </filter>  
  """  
  loopback = {"int\_name": "Loopback102",  
   "description": "Demo interface by NETCONF",  
   "ip": "192.168.102.1",  
   "netmask": "255.255.255.0"}  
  m = manager.connect(host = device["address"],  
   port = device["netconf\_port"],  
   username = device["username"],  
   password = device["password"],  
   hostkey\_verify = False)

1. Verify NETCONF connection is active

* m.connected

1. Create new config template to delete an interface

* config\_data = """  
  <config>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface operation="delete">  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </config>  
  """

1. Create desired NETCONF config payload and execute to delete the interface

* config = config\_data.format(\*\*loopback)  
  r = m.edit\_config(target = "running", config = config)

1. Print OK status (expected output true)

* print("NETCONF RPC OK: {}".format(r.ok))

1. Create a new NETCONF to check on new loopback interface

* filter = interface\_filter.format(int\_name = "Loopback102")

1. Execute a NETCONF using this filter

* r = m.get\_config("running", filter)

1. Pretty print the raw XML to screen (expected output will not include the loopback interface, as you just deleted it)

* xml\_doc = minidom.parseString(r.xml)  
  print(xml\_doc.toprettyxml(indent = " "))

### End the NETCONF Connection

1. Send a RPC request to disconnect the connection.

* m.close\_session()  
  m.connected

1. End the Python interpreter.

* exit()

## CLI with netmiko

1. From the root of the python\_networking repository, change into the exercise directory.

* cd device\_apis/cli

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

### Retrieve Network Configuration Details with CLI with netmiko\_example1.py

1. Import libraries

* from netmiko import ConnectHandler  
  import re  
  import sys

1. Add parent directory to path to allow importing common vars

* sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device

1. Set device\_type for netmiko

* device["device\_type"] = "cisco\_ios"

1. Create a CLI command template

* show\_interface\_config\_temp = "show running-config interface {}"

1. Open CLI connection to device.
   * *Note: Normally you’d use a with block to open connection to device. This avoids needing to manually m.close\_session() at the end of a script, but for interactive use, this format is chosen.*

* ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create desired CLI command

* command = show\_interface\_config\_temp.format("GigabitEthernet2")

1. Verify the command has been created correctly

* print(command)

1. Send command to device

* interface = ch.send\_command(command)

1. Print the raw command output to the screen

* print(interface)

1. Create regular expression searches to parse the output for desired interface details

* name = re.search(r'interface (.\*)', interface).group(1)  
  description = re.search(r'description (.\*)', interface).group(1)

1. Pull out the ip and mask for the interface

* ip\_info = re.search(r'ip address (.\*) (.\*)', interface)  
  ip = ip\_info.group(1)  
  netmask = ip\_info.group(2)

1. Print the desired info to the screen

* print("The interface {name} has ip address {ip}/{mask}".format(  
   name = name,  
   ip = ip,  
   mask = netmask,  
   )  
   )

### Modify Network Configuration Details with CLI with netmiko\_example2.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from netmiko import ConnectHandler  
  import re, sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  device["device\_type"] = "cisco\_ios"  
  show\_interface\_config\_temp = "show running-config interface {}"  
  ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create Python dictionary with new Loopback Details

* loopback = {"int\_name": "Loopback103",  
   "description": "Demo interface by CLI and netmiko",  
   "ip": "192.168.103.1",  
   "netmask": "255.255.255.0"}

1. Create a CLI configuration

* interface\_config = [  
   "interface {}".format(loopback["int\_name"]),  
   "description {}".format(loopback["description"]),  
   "ip address {} {}".format(loopback["ip"], loopback["netmask"]),  
   "no shut"  
  ]

1. Send configuration to device

* output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to retrieve the new configuration.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)

### Delete Network Configuration Details with CLI with netmiko\_example3.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from netmiko import ConnectHandler  
  import re, sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  device["device\_type"] = "cisco\_ios"  
  show\_interface\_config\_temp = "show running-config interface {}"  
  ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create a new CLI configuration to delete the interface.

* interface\_config = [  
   "no interface {}".format(loopback["int\_name"])  
  ]

1. Send configuration to device

* output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to verify configuration removed.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)
* *Note: attempting to view the configuration of a non-existing interface will generate a CLI error. This output is expected, and one of the reasons APIs like NETCONF or RESTCONF are better suited to programmatic interactions.*

### End the CLI connection to the device

1. Disconnect from the device.

* ch.disconnect()

1. End the Python interpreter.

* exit()

## Other Cool Python Stuff

### Introduction to pyATS

[pyATS](https://developer.cisco.com/site/pyats) is a network testing tool developed by Cisco and made available for free, with significant elements of the underlying code open source.

pyATS offers network developers the ability to profile the network state of hardware, interfaces, protocols, etc… before, during and after changes, to ensure the network is operating as designed, and identify problems before the dreaded phone call. To enable this level of robust testing, pyATS offers a standard way to communicate with network elements and standardize the data returned into native Python objects. This core functionality opens up a lot of flexibility on how pyATS can be used by network developers.

In the following exercises, you will get a brief introduction to pyATS to connect and learn about device details.

### Connect and Interact with a Device

1. From the root of the python\_networking repository, change into the exercise directory.

* cd network\_testing/pyats

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

1. Import in pyATS libraries and tools

* from genie.conf import Genie  
  from ats.topology import loader  
  from genie.abstract import Lookup  
  from genie.libs import ops # noqa

1. Read and process the testbed (inventory) file

* genie\_testbed = Genie.init("./default\_testbed.yaml")

1. Create a pyATS device object from testbed

* vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()

1. Query for details on the new interface (no output expected, as you just deleted it)

* r = requests.get(url,  
   headers = restconf\_headers,  
   auth=(device["username"], device["password"]),  
   verify=False)

1. Check status code (expected 404)

* print(r.status\_code)

## NETCONF with ncclient

1. From the root of the python\_networking repository, change into the exercise directory.

* cd device\_apis/netconf

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

### Retrieve Network Configuration Details with NETCONF with netconf\_example1.py

1. Import libraries

* from ncclient import manager  
  from xml.dom import minidom  
  import xmltodict  
  import sys

1. Add parent directory to path to allow importing common vars

* sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device

1. Create filter template for an interface

* interface\_filter = """  
  <filter>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </filter>  
  """

1. Open NETCONF connection to device
   * *Note: Normally you’d use a with block to open connection to device. This avoids needing to manually m.close\_session() at the end of a script, but for interactive use, this format is chosen.*

* m = manager.connect(host = device["address"],  
   port = device["netconf\_port"],  
   username = device["username"],  
   password = device["password"],  
   hostkey\_verify = False)

1. Verify NETCONF connection is active (expected output true)

* m.connected

1. Create desired NETCONF filter for a particular interface

* filter = interface\_filter.format(int\_name = "GigabitEthernet2")

1. Execute a NETCONF using the filter

* r = m.get\_config("running", filter)

1. Pretty print raw xml to screen

* xml\_doc = minidom.parseString(r.xml)  
  print(xml\_doc.toprettyxml(indent = " "))

1. Process the XML data into Python Dictionary and use

* interface = xmltodict.parse(r.xml)

1. Pretty Print the full Python (Ordered) Dictionary.

* from pprint import pprint  
  pprint(interface)

1. If RPC returned data, print out the interesting pieces.

* if not interface["rpc-reply"]["data"] is None:  
   # Create Python variable for interface details  
   interface = interface["rpc-reply"]["data"]["interfaces"]["interface"]  
    
   print("The interface {name} has ip address {ip}/{mask}".format(  
   name = interface["name"]["#text"],  
   ip = interface["ipv4"]["address"]["ip"],  
   mask = interface["ipv4"]["address"]["netmask"],  
   )  
   )  
  else:  
   print("No interface {} found".format("GigabitEthernet2"))

### Modify Network Configuration Details with NETCONF with netconf\_example2.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from ncclient import manager  
  from xml.dom import minidom  
  import xmltodict  
  import sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  interface\_filter = """  
  <filter>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </filter>  
  """  
  m = manager.connect(host = device["address"],  
   port = device["netconf\_port"],  
   username = device["username"],  
   password = device["password"],  
   hostkey\_verify = False)

1. Verify NETCONF connection is active

* m.connected

1. Create Python dictionary with new Loopback Details

* loopback = {"int\_name": "Loopback102",  
   "description": "Demo interface by NETCONF",  
   "ip": "192.168.102.1",  
   "netmask": "255.255.255.0"}

1. Create NETCONF template for an interface

* config\_data = """  
  <config>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   <description>{description}</description>  
   <type xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">  
   ianaift:softwareLoopback  
   </type>  
   <enabled>true</enabled>  
   <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">  
   <address>  
   <ip>{ip}</ip>  
   <netmask>{netmask}</netmask>  
   </address>  
   </ipv4>  
   </interface>  
   </interfaces>  
  </config>  
  """

1. Create desired NETCONF config payload

* config = config\_data.format(\*\*loopback)

1. Send operation

* r = m.edit\_config(target = "running", config = config)

1. Print OK status (expected output true)

* print("NETCONF RPC OK: {}".format(r.ok))

1. Create a new NETCONF to check on new loopback interface

* filter = interface\_filter.format(int\_name = "Loopback102")

1. Execute a NETCONF using this filter

* r = m.get\_config("running", filter)

1. Pretty print the raw XML to screen

* xml\_doc = minidom.parseString(r.xml)  
  print(xml\_doc.toprettyxml(indent = " "))

### Delete Network Configuration Details with NETCONF with netconf\_example3.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from ncclient import manager  
  from xml.dom import minidom  
  import xmltodict  
  import sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  interface\_filter = """  
  <filter>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </filter>  
  """  
  loopback = {"int\_name": "Loopback102",  
   "description": "Demo interface by NETCONF",  
   "ip": "192.168.102.1",  
   "netmask": "255.255.255.0"}  
  m = manager.connect(host = device["address"],  
   port = device["netconf\_port"],  
   username = device["username"],  
   password = device["password"],  
   hostkey\_verify = False)

1. Verify NETCONF connection is active

* m.connected

1. Create new config template to delete an interface

* config\_data = """  
  <config>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface operation="delete">  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </config>  
  """

1. Create desired NETCONF config payload and execute to delete the interface

* config = config\_data.format(\*\*loopback)  
  r = m.edit\_config(target = "running", config = config)

1. Print OK status (expected output true)

* print("NETCONF RPC OK: {}".format(r.ok))

1. Create a new NETCONF to check on new loopback interface

* filter = interface\_filter.format(int\_name = "Loopback102")

1. Execute a NETCONF using this filter

* r = m.get\_config("running", filter)

1. Pretty print the raw XML to screen (expected output will not include the loopback interface, as you just deleted it)

* xml\_doc = minidom.parseString(r.xml)  
  print(xml\_doc.toprettyxml(indent = " "))

### End the NETCONF Connection

1. Send a RPC request to disconnect the connection.

* m.close\_session()  
  m.connected

1. End the Python interpreter.

* exit()

## CLI with netmiko

1. From the root of the python\_networking repository, change into the exercise directory.

* cd device\_apis/cli

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

### Retrieve Network Configuration Details with CLI with netmiko\_example1.py

1. Import libraries

* from netmiko import ConnectHandler  
  import re  
  import sys

1. Add parent directory to path to allow importing common vars

* sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device

1. Set device\_type for netmiko

* device["device\_type"] = "cisco\_ios"

1. Create a CLI command template

* show\_interface\_config\_temp = "show running-config interface {}"

1. Open CLI connection to device.
   * *Note: Normally you’d use a with block to open connection to device. This avoids needing to manually m.close\_session() at the end of a script, but for interactive use, this format is chosen.*

* ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create desired CLI command

* command = show\_interface\_config\_temp.format("GigabitEthernet2")

1. Verify the command has been created correctly

* print(command)

1. Send command to device

* interface = ch.send\_command(command)

1. Print the raw command output to the screen

* print(interface)

1. Create regular expression searches to parse the output for desired interface details

* name = re.search(r'interface (.\*)', interface).group(1)  
  description = re.search(r'description (.\*)', interface).group(1)

1. Pull out the ip and mask for the interface

* ip\_info = re.search(r'ip address (.\*) (.\*)', interface)  
  ip = ip\_info.group(1)  
  netmask = ip\_info.group(2)

1. Print the desired info to the screen

* print("The interface {name} has ip address {ip}/{mask}".format(  
   name = name,  
   ip = ip,  
   mask = netmask,  
   )  
   )

### Modify Network Configuration Details with CLI with netmiko\_example2.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from netmiko import ConnectHandler  
  import re, sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  device["device\_type"] = "cisco\_ios"  
  show\_interface\_config\_temp = "show running-config interface {}"  
  ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create Python dictionary with new Loopback Details

* loopback = {"int\_name": "Loopback103",  
   "description": "Demo interface by CLI and netmiko",  
   "ip": "192.168.103.1",  
   "netmask": "255.255.255.0"}

1. Create a CLI configuration

* interface\_config = [  
   "interface {}".format(loopback["int\_name"]),  
   "description {}".format(loopback["description"]),  
   "ip address {} {}".format(loopback["ip"], loopback["netmask"]),  
   "no shut"  
  ]

1. Send configuration to device

* output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to retrieve the new configuration.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)

### Delete Network Configuration Details with CLI with netmiko\_example3.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from netmiko import ConnectHandler  
  import re, sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  device["device\_type"] = "cisco\_ios"  
  show\_interface\_config\_temp = "show running-config interface {}"  
  ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create a new CLI configuration to delete the interface.

* interface\_config = [  
   "no interface {}".format(loopback["int\_name"])  
  ]

1. Send configuration to device

* output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to verify configuration removed.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)
* *Note: attempting to view the configuration of a non-existing interface will generate a CLI error. This output is expected, and one of the reasons APIs like NETCONF or RESTCONF are better suited to programmatic interactions.*

### End the CLI connection to the device

1. Disconnect from the device.

* ch.disconnect()

1. End the Python interpreter.

* exit()

## Other Cool Python Stuff

### Introduction to pyATS

[pyATS](https://developer.cisco.com/site/pyats) is a network testing tool developed by Cisco and made available for free, with significant elements of the underlying code open source.

pyATS offers network developers the ability to profile the network state of hardware, interfaces, protocols, etc… before, during and after changes, to ensure the network is operating as designed, and identify problems before the dreaded phone call. To enable this level of robust testing, pyATS offers a standard way to communicate with network elements and standardize the data returned into native Python objects. This core functionality opens up a lot of flexibility on how pyATS can be used by network developers.

In the following exercises, you will get a brief introduction to pyATS to connect and learn about device details.

### Connect and Interact with a Device

1. From the root of the python\_networking repository, change into the exercise directory.

* cd network\_testing/pyats

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

1. Import in pyATS libraries and tools

* from genie.conf import Genie  
  from ats.topology import loader  
  from genie.abstract import Lookup  
  from genie.libs import ops # noqa

1. Read and process the testbed (inventory) file

* genie\_testbed = Genie.init("./default\_testbed.yaml")

1. Create a pyATS device object from testbed

* vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()
* print("Request Status Code: {}".format(r.status\_code))

1. Query for details on the new interface (no output expected, as you just deleted it)

* r = requests.get(url,  
   headers = restconf\_headers,  
   auth=(device["username"], device["password"]),  
   verify=False)

1. Check status code (expected 404)

* print(r.status\_code)

## NETCONF with ncclient

1. From the root of the python\_networking repository, change into the exercise directory.

* cd device\_apis/netconf

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

### Retrieve Network Configuration Details with NETCONF with netconf\_example1.py

1. Import libraries

* from ncclient import manager  
  from xml.dom import minidom  
  import xmltodict  
  import sys

1. Add parent directory to path to allow importing common vars

* sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device

1. Create filter template for an interface

* interface\_filter = """  
  <filter>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </filter>  
  """

1. Open NETCONF connection to device
   * *Note: Normally you’d use a with block to open connection to device. This avoids needing to manually m.close\_session() at the end of a script, but for interactive use, this format is chosen.*

* m = manager.connect(host = device["address"],  
   port = device["netconf\_port"],  
   username = device["username"],  
   password = device["password"],  
   hostkey\_verify = False)

1. Verify NETCONF connection is active (expected output true)

* m.connected

1. Create desired NETCONF filter for a particular interface

* filter = interface\_filter.format(int\_name = "GigabitEthernet2")

1. Execute a NETCONF using the filter

* r = m.get\_config("running", filter)

1. Pretty print raw xml to screen

* xml\_doc = minidom.parseString(r.xml)  
  print(xml\_doc.toprettyxml(indent = " "))

1. Process the XML data into Python Dictionary and use

* interface = xmltodict.parse(r.xml)

1. Pretty Print the full Python (Ordered) Dictionary.

* from pprint import pprint  
  pprint(interface)

1. If RPC returned data, print out the interesting pieces.

* if not interface["rpc-reply"]["data"] is None:  
   # Create Python variable for interface details  
   interface = interface["rpc-reply"]["data"]["interfaces"]["interface"]  
    
   print("The interface {name} has ip address {ip}/{mask}".format(  
   name = interface["name"]["#text"],  
   ip = interface["ipv4"]["address"]["ip"],  
   mask = interface["ipv4"]["address"]["netmask"],  
   )  
   )  
  else:  
   print("No interface {} found".format("GigabitEthernet2"))

### Modify Network Configuration Details with NETCONF with netconf\_example2.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from ncclient import manager  
  from xml.dom import minidom  
  import xmltodict  
  import sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  interface\_filter = """  
  <filter>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </filter>  
  """  
  m = manager.connect(host = device["address"],  
   port = device["netconf\_port"],  
   username = device["username"],  
   password = device["password"],  
   hostkey\_verify = False)

1. Verify NETCONF connection is active

* m.connected

1. Create Python dictionary with new Loopback Details

* loopback = {"int\_name": "Loopback102",  
   "description": "Demo interface by NETCONF",  
   "ip": "192.168.102.1",  
   "netmask": "255.255.255.0"}

1. Create NETCONF template for an interface

* config\_data = """  
  <config>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   <description>{description}</description>  
   <type xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">  
   ianaift:softwareLoopback  
   </type>  
   <enabled>true</enabled>  
   <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">  
   <address>  
   <ip>{ip}</ip>  
   <netmask>{netmask}</netmask>  
   </address>  
   </ipv4>  
   </interface>  
   </interfaces>  
  </config>  
  """

1. Create desired NETCONF config payload

* config = config\_data.format(\*\*loopback)

1. Send operation

* r = m.edit\_config(target = "running", config = config)

1. Print OK status (expected output true)

* print("NETCONF RPC OK: {}".format(r.ok))

1. Create a new NETCONF to check on new loopback interface

* filter = interface\_filter.format(int\_name = "Loopback102")

1. Execute a NETCONF using this filter

* r = m.get\_config("running", filter)

1. Pretty print the raw XML to screen

* xml\_doc = minidom.parseString(r.xml)  
  print(xml\_doc.toprettyxml(indent = " "))

### Delete Network Configuration Details with NETCONF with netconf\_example3.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from ncclient import manager  
  from xml.dom import minidom  
  import xmltodict  
  import sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  interface\_filter = """  
  <filter>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </filter>  
  """  
  loopback = {"int\_name": "Loopback102",  
   "description": "Demo interface by NETCONF",  
   "ip": "192.168.102.1",  
   "netmask": "255.255.255.0"}  
  m = manager.connect(host = device["address"],  
   port = device["netconf\_port"],  
   username = device["username"],  
   password = device["password"],  
   hostkey\_verify = False)

1. Verify NETCONF connection is active

* m.connected

1. Create new config template to delete an interface

* config\_data = """  
  <config>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface operation="delete">  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </config>  
  """

1. Create desired NETCONF config payload and execute to delete the interface

* config = config\_data.format(\*\*loopback)  
  r = m.edit\_config(target = "running", config = config)

1. Print OK status (expected output true)

* print("NETCONF RPC OK: {}".format(r.ok))

1. Create a new NETCONF to check on new loopback interface

* filter = interface\_filter.format(int\_name = "Loopback102")

1. Execute a NETCONF using this filter

* r = m.get\_config("running", filter)

1. Pretty print the raw XML to screen (expected output will not include the loopback interface, as you just deleted it)

* xml\_doc = minidom.parseString(r.xml)  
  print(xml\_doc.toprettyxml(indent = " "))

### End the NETCONF Connection

1. Send a RPC request to disconnect the connection.

* m.close\_session()  
  m.connected

1. End the Python interpreter.

* exit()

## CLI with netmiko

1. From the root of the python\_networking repository, change into the exercise directory.

* cd device\_apis/cli

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

### Retrieve Network Configuration Details with CLI with netmiko\_example1.py

1. Import libraries

* from netmiko import ConnectHandler  
  import re  
  import sys

1. Add parent directory to path to allow importing common vars

* sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device

1. Set device\_type for netmiko

* device["device\_type"] = "cisco\_ios"

1. Create a CLI command template

* show\_interface\_config\_temp = "show running-config interface {}"

1. Open CLI connection to device.
   * *Note: Normally you’d use a with block to open connection to device. This avoids needing to manually m.close\_session() at the end of a script, but for interactive use, this format is chosen.*

* ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create desired CLI command

* command = show\_interface\_config\_temp.format("GigabitEthernet2")

1. Verify the command has been created correctly

* print(command)

1. Send command to device

* interface = ch.send\_command(command)

1. Print the raw command output to the screen

* print(interface)

1. Create regular expression searches to parse the output for desired interface details

* name = re.search(r'interface (.\*)', interface).group(1)  
  description = re.search(r'description (.\*)', interface).group(1)

1. Pull out the ip and mask for the interface

* ip\_info = re.search(r'ip address (.\*) (.\*)', interface)  
  ip = ip\_info.group(1)  
  netmask = ip\_info.group(2)

1. Print the desired info to the screen

* print("The interface {name} has ip address {ip}/{mask}".format(  
   name = name,  
   ip = ip,  
   mask = netmask,  
   )  
   )

### Modify Network Configuration Details with CLI with netmiko\_example2.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from netmiko import ConnectHandler  
  import re, sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  device["device\_type"] = "cisco\_ios"  
  show\_interface\_config\_temp = "show running-config interface {}"  
  ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create Python dictionary with new Loopback Details

* loopback = {"int\_name": "Loopback103",  
   "description": "Demo interface by CLI and netmiko",  
   "ip": "192.168.103.1",  
   "netmask": "255.255.255.0"}

1. Create a CLI configuration

* interface\_config = [  
   "interface {}".format(loopback["int\_name"]),  
   "description {}".format(loopback["description"]),  
   "ip address {} {}".format(loopback["ip"], loopback["netmask"]),  
   "no shut"  
  ]

1. Send configuration to device

* output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to retrieve the new configuration.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)

### Delete Network Configuration Details with CLI with netmiko\_example3.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from netmiko import ConnectHandler  
  import re, sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  device["device\_type"] = "cisco\_ios"  
  show\_interface\_config\_temp = "show running-config interface {}"  
  ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create a new CLI configuration to delete the interface.

* interface\_config = [  
   "no interface {}".format(loopback["int\_name"])  
  ]

1. Send configuration to device

* output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to verify configuration removed.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)
* *Note: attempting to view the configuration of a non-existing interface will generate a CLI error. This output is expected, and one of the reasons APIs like NETCONF or RESTCONF are better suited to programmatic interactions.*

### End the CLI connection to the device

1. Disconnect from the device.

* ch.disconnect()

1. End the Python interpreter.

* exit()

## Other Cool Python Stuff

### Introduction to pyATS

[pyATS](https://developer.cisco.com/site/pyats) is a network testing tool developed by Cisco and made available for free, with significant elements of the underlying code open source.

pyATS offers network developers the ability to profile the network state of hardware, interfaces, protocols, etc… before, during and after changes, to ensure the network is operating as designed, and identify problems before the dreaded phone call. To enable this level of robust testing, pyATS offers a standard way to communicate with network elements and standardize the data returned into native Python objects. This core functionality opens up a lot of flexibility on how pyATS can be used by network developers.

In the following exercises, you will get a brief introduction to pyATS to connect and learn about device details.

### Connect and Interact with a Device

1. From the root of the python\_networking repository, change into the exercise directory.

* cd network\_testing/pyats

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

1. Import in pyATS libraries and tools

* from genie.conf import Genie  
  from ats.topology import loader  
  from genie.abstract import Lookup  
  from genie.libs import ops # noqa

1. Read and process the testbed (inventory) file

* genie\_testbed = Genie.init("./default\_testbed.yaml")

1. Create a pyATS device object from testbed

* vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()

### Delete Network Configuration Details with RESTCONF with restconf\_example3.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* import requests, urllib3, sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  urllib3.disable\_warnings(urllib3.exceptions.InsecureRequestWarning)  
  restconf\_headers = {"Accept": "application/yang-data+json"}  
  restconf\_base = "https://{ip}:{port}/restconf/data"  
  interface\_url = restconf\_base + "/ietf-interfaces:interfaces/interface={int\_name}"  
  url = interface\_url.format(ip = device["address"],  
   port = device["restconf\_port"],  
   int\_name = "Loopback101"  
   )

1. Send DELETE request to remove the Loopback.

* r = requests.delete(url,  
   headers = restconf\_headers,  
   auth=(device["username"], device["password"]),  
   verify=False)

1. Check Status Code (expected 204)

* print("Request Status Code: {}".format(r.status\_code))

1. Query for details on the new interface (no output expected, as you just deleted it)

* r = requests.get(url,  
   headers = restconf\_headers,  
   auth=(device["username"], device["password"]),  
   verify=False)

1. Check status code (expected 404)

* print(r.status\_code)

## NETCONF with ncclient

1. From the root of the python\_networking repository, change into the exercise directory.

* cd device\_apis/netconf

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

### Retrieve Network Configuration Details with NETCONF with netconf\_example1.py

1. Import libraries

* from ncclient import manager  
  from xml.dom import minidom  
  import xmltodict  
  import sys

1. Add parent directory to path to allow importing common vars

* sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device

1. Create filter template for an interface

* interface\_filter = """  
  <filter>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </filter>  
  """

1. Open NETCONF connection to device
   * *Note: Normally you’d use a with block to open connection to device. This avoids needing to manually m.close\_session() at the end of a script, but for interactive use, this format is chosen.*

* m = manager.connect(host = device["address"],  
   port = device["netconf\_port"],  
   username = device["username"],  
   password = device["password"],  
   hostkey\_verify = False)

1. Verify NETCONF connection is active (expected output true)

* m.connected

1. Create desired NETCONF filter for a particular interface

* filter = interface\_filter.format(int\_name = "GigabitEthernet2")

1. Execute a NETCONF using the filter

* r = m.get\_config("running", filter)

1. Pretty print raw xml to screen

* xml\_doc = minidom.parseString(r.xml)  
  print(xml\_doc.toprettyxml(indent = " "))

1. Process the XML data into Python Dictionary and use

* interface = xmltodict.parse(r.xml)

1. Pretty Print the full Python (Ordered) Dictionary.

* from pprint import pprint  
  pprint(interface)

1. If RPC returned data, print out the interesting pieces.

* if not interface["rpc-reply"]["data"] is None:  
   # Create Python variable for interface details  
   interface = interface["rpc-reply"]["data"]["interfaces"]["interface"]  
    
   print("The interface {name} has ip address {ip}/{mask}".format(  
   name = interface["name"]["#text"],  
   ip = interface["ipv4"]["address"]["ip"],  
   mask = interface["ipv4"]["address"]["netmask"],  
   )  
   )  
  else:  
   print("No interface {} found".format("GigabitEthernet2"))

### Modify Network Configuration Details with NETCONF with netconf\_example2.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from ncclient import manager  
  from xml.dom import minidom  
  import xmltodict  
  import sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  interface\_filter = """  
  <filter>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </filter>  
  """  
  m = manager.connect(host = device["address"],  
   port = device["netconf\_port"],  
   username = device["username"],  
   password = device["password"],  
   hostkey\_verify = False)

1. Verify NETCONF connection is active

* m.connected

1. Create Python dictionary with new Loopback Details

* loopback = {"int\_name": "Loopback102",  
   "description": "Demo interface by NETCONF",  
   "ip": "192.168.102.1",  
   "netmask": "255.255.255.0"}

1. Create NETCONF template for an interface

* config\_data = """  
  <config>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   <description>{description}</description>  
   <type xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">  
   ianaift:softwareLoopback  
   </type>  
   <enabled>true</enabled>  
   <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">  
   <address>  
   <ip>{ip}</ip>  
   <netmask>{netmask}</netmask>  
   </address>  
   </ipv4>  
   </interface>  
   </interfaces>  
  </config>  
  """

1. Create desired NETCONF config payload

* config = config\_data.format(\*\*loopback)

1. Send operation

* r = m.edit\_config(target = "running", config = config)

1. Print OK status (expected output true)

* print("NETCONF RPC OK: {}".format(r.ok))

1. Create a new NETCONF to check on new loopback interface

* filter = interface\_filter.format(int\_name = "Loopback102")

1. Execute a NETCONF using this filter

* r = m.get\_config("running", filter)

1. Pretty print the raw XML to screen

* xml\_doc = minidom.parseString(r.xml)  
  print(xml\_doc.toprettyxml(indent = " "))

### Delete Network Configuration Details with NETCONF with netconf\_example3.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from ncclient import manager  
  from xml.dom import minidom  
  import xmltodict  
  import sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  interface\_filter = """  
  <filter>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </filter>  
  """  
  loopback = {"int\_name": "Loopback102",  
   "description": "Demo interface by NETCONF",  
   "ip": "192.168.102.1",  
   "netmask": "255.255.255.0"}  
  m = manager.connect(host = device["address"],  
   port = device["netconf\_port"],  
   username = device["username"],  
   password = device["password"],  
   hostkey\_verify = False)

1. Verify NETCONF connection is active

* m.connected

1. Create new config template to delete an interface

* config\_data = """  
  <config>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface operation="delete">  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </config>  
  """

1. Create desired NETCONF config payload and execute to delete the interface

* config = config\_data.format(\*\*loopback)  
  r = m.edit\_config(target = "running", config = config)

1. Print OK status (expected output true)

* print("NETCONF RPC OK: {}".format(r.ok))

1. Create a new NETCONF to check on new loopback interface

* filter = interface\_filter.format(int\_name = "Loopback102")

1. Execute a NETCONF using this filter

* r = m.get\_config("running", filter)

1. Pretty print the raw XML to screen (expected output will not include the loopback interface, as you just deleted it)

* xml\_doc = minidom.parseString(r.xml)  
  print(xml\_doc.toprettyxml(indent = " "))

### End the NETCONF Connection

1. Send a RPC request to disconnect the connection.

* m.close\_session()  
  m.connected

1. End the Python interpreter.

* exit()

## CLI with netmiko

1. From the root of the python\_networking repository, change into the exercise directory.

* cd device\_apis/cli

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

### Retrieve Network Configuration Details with CLI with netmiko\_example1.py

1. Import libraries

* from netmiko import ConnectHandler  
  import re  
  import sys

1. Add parent directory to path to allow importing common vars

* sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device

1. Set device\_type for netmiko

* device["device\_type"] = "cisco\_ios"

1. Create a CLI command template

* show\_interface\_config\_temp = "show running-config interface {}"

1. Open CLI connection to device.
   * *Note: Normally you’d use a with block to open connection to device. This avoids needing to manually m.close\_session() at the end of a script, but for interactive use, this format is chosen.*

* ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create desired CLI command

* command = show\_interface\_config\_temp.format("GigabitEthernet2")

1. Verify the command has been created correctly

* print(command)

1. Send command to device

* interface = ch.send\_command(command)

1. Print the raw command output to the screen

* print(interface)

1. Create regular expression searches to parse the output for desired interface details

* name = re.search(r'interface (.\*)', interface).group(1)  
  description = re.search(r'description (.\*)', interface).group(1)

1. Pull out the ip and mask for the interface

* ip\_info = re.search(r'ip address (.\*) (.\*)', interface)  
  ip = ip\_info.group(1)  
  netmask = ip\_info.group(2)

1. Print the desired info to the screen

* print("The interface {name} has ip address {ip}/{mask}".format(  
   name = name,  
   ip = ip,  
   mask = netmask,  
   )  
   )

### Modify Network Configuration Details with CLI with netmiko\_example2.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from netmiko import ConnectHandler  
  import re, sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  device["device\_type"] = "cisco\_ios"  
  show\_interface\_config\_temp = "show running-config interface {}"  
  ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create Python dictionary with new Loopback Details

* loopback = {"int\_name": "Loopback103",  
   "description": "Demo interface by CLI and netmiko",  
   "ip": "192.168.103.1",  
   "netmask": "255.255.255.0"}

1. Create a CLI configuration

* interface\_config = [  
   "interface {}".format(loopback["int\_name"]),  
   "description {}".format(loopback["description"]),  
   "ip address {} {}".format(loopback["ip"], loopback["netmask"]),  
   "no shut"  
  ]

1. Send configuration to device

* output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to retrieve the new configuration.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)

### Delete Network Configuration Details with CLI with netmiko\_example3.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from netmiko import ConnectHandler  
  import re, sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  device["device\_type"] = "cisco\_ios"  
  show\_interface\_config\_temp = "show running-config interface {}"  
  ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create a new CLI configuration to delete the interface.

* interface\_config = [  
   "no interface {}".format(loopback["int\_name"])  
  ]

1. Send configuration to device

* output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to verify configuration removed.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)
* *Note: attempting to view the configuration of a non-existing interface will generate a CLI error. This output is expected, and one of the reasons APIs like NETCONF or RESTCONF are better suited to programmatic interactions.*

### End the CLI connection to the device

1. Disconnect from the device.

* ch.disconnect()

1. End the Python interpreter.

* exit()

## Other Cool Python Stuff

### Introduction to pyATS

[pyATS](https://developer.cisco.com/site/pyats) is a network testing tool developed by Cisco and made available for free, with significant elements of the underlying code open source.

pyATS offers network developers the ability to profile the network state of hardware, interfaces, protocols, etc… before, during and after changes, to ensure the network is operating as designed, and identify problems before the dreaded phone call. To enable this level of robust testing, pyATS offers a standard way to communicate with network elements and standardize the data returned into native Python objects. This core functionality opens up a lot of flexibility on how pyATS can be used by network developers.

In the following exercises, you will get a brief introduction to pyATS to connect and learn about device details.

### Connect and Interact with a Device

1. From the root of the python\_networking repository, change into the exercise directory.

* cd network\_testing/pyats

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

1. Import in pyATS libraries and tools

* from genie.conf import Genie  
  from ats.topology import loader  
  from genie.abstract import Lookup  
  from genie.libs import ops # noqa

1. Read and process the testbed (inventory) file

* genie\_testbed = Genie.init("./default\_testbed.yaml")

1. Create a pyATS device object from testbed

* vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()
* # Create URL and send RESTCONF request to core1 for GigE2 Config  
  url = interface\_url.format(ip = device["address"],  
   port = device["restconf\_port"],  
   int\_name = "Loopback101"  
   )  
  r = requests.get(url,  
   headers = restconf\_headers,  
   auth=(device["username"], device["password"]),  
   verify=False)  
    
  # Print returned data  
  print(r.text)

### Delete Network Configuration Details with RESTCONF with restconf\_example3.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* import requests, urllib3, sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  urllib3.disable\_warnings(urllib3.exceptions.InsecureRequestWarning)  
  restconf\_headers = {"Accept": "application/yang-data+json"}  
  restconf\_base = "https://{ip}:{port}/restconf/data"  
  interface\_url = restconf\_base + "/ietf-interfaces:interfaces/interface={int\_name}"  
  url = interface\_url.format(ip = device["address"],  
   port = device["restconf\_port"],  
   int\_name = "Loopback101"  
   )

1. Send DELETE request to remove the Loopback.

* r = requests.delete(url,  
   headers = restconf\_headers,  
   auth=(device["username"], device["password"]),  
   verify=False)

1. Check Status Code (expected 204)

* print("Request Status Code: {}".format(r.status\_code))

1. Query for details on the new interface (no output expected, as you just deleted it)

* r = requests.get(url,  
   headers = restconf\_headers,  
   auth=(device["username"], device["password"]),  
   verify=False)

1. Check status code (expected 404)

* print(r.status\_code)

## NETCONF with ncclient

1. From the root of the python\_networking repository, change into the exercise directory.

* cd device\_apis/netconf

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

### Retrieve Network Configuration Details with NETCONF with netconf\_example1.py

1. Import libraries

* from ncclient import manager  
  from xml.dom import minidom  
  import xmltodict  
  import sys

1. Add parent directory to path to allow importing common vars

* sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device

1. Create filter template for an interface

* interface\_filter = """  
  <filter>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </filter>  
  """

1. Open NETCONF connection to device
   * *Note: Normally you’d use a with block to open connection to device. This avoids needing to manually m.close\_session() at the end of a script, but for interactive use, this format is chosen.*

* m = manager.connect(host = device["address"],  
   port = device["netconf\_port"],  
   username = device["username"],  
   password = device["password"],  
   hostkey\_verify = False)

1. Verify NETCONF connection is active (expected output true)

* m.connected

1. Create desired NETCONF filter for a particular interface

* filter = interface\_filter.format(int\_name = "GigabitEthernet2")

1. Execute a NETCONF using the filter

* r = m.get\_config("running", filter)

1. Pretty print raw xml to screen

* xml\_doc = minidom.parseString(r.xml)  
  print(xml\_doc.toprettyxml(indent = " "))

1. Process the XML data into Python Dictionary and use

* interface = xmltodict.parse(r.xml)

1. Pretty Print the full Python (Ordered) Dictionary.

* from pprint import pprint  
  pprint(interface)

1. If RPC returned data, print out the interesting pieces.

* if not interface["rpc-reply"]["data"] is None:  
   # Create Python variable for interface details  
   interface = interface["rpc-reply"]["data"]["interfaces"]["interface"]  
    
   print("The interface {name} has ip address {ip}/{mask}".format(  
   name = interface["name"]["#text"],  
   ip = interface["ipv4"]["address"]["ip"],  
   mask = interface["ipv4"]["address"]["netmask"],  
   )  
   )  
  else:  
   print("No interface {} found".format("GigabitEthernet2"))

### Modify Network Configuration Details with NETCONF with netconf\_example2.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from ncclient import manager  
  from xml.dom import minidom  
  import xmltodict  
  import sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  interface\_filter = """  
  <filter>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </filter>  
  """  
  m = manager.connect(host = device["address"],  
   port = device["netconf\_port"],  
   username = device["username"],  
   password = device["password"],  
   hostkey\_verify = False)

1. Verify NETCONF connection is active

* m.connected

1. Create Python dictionary with new Loopback Details

* loopback = {"int\_name": "Loopback102",  
   "description": "Demo interface by NETCONF",  
   "ip": "192.168.102.1",  
   "netmask": "255.255.255.0"}

1. Create NETCONF template for an interface

* config\_data = """  
  <config>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   <description>{description}</description>  
   <type xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">  
   ianaift:softwareLoopback  
   </type>  
   <enabled>true</enabled>  
   <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">  
   <address>  
   <ip>{ip}</ip>  
   <netmask>{netmask}</netmask>  
   </address>  
   </ipv4>  
   </interface>  
   </interfaces>  
  </config>  
  """

1. Create desired NETCONF config payload

* config = config\_data.format(\*\*loopback)

1. Send operation

* r = m.edit\_config(target = "running", config = config)

1. Print OK status (expected output true)

* print("NETCONF RPC OK: {}".format(r.ok))

1. Create a new NETCONF to check on new loopback interface

* filter = interface\_filter.format(int\_name = "Loopback102")

1. Execute a NETCONF using this filter

* r = m.get\_config("running", filter)

1. Pretty print the raw XML to screen

* xml\_doc = minidom.parseString(r.xml)  
  print(xml\_doc.toprettyxml(indent = " "))

### Delete Network Configuration Details with NETCONF with netconf\_example3.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from ncclient import manager  
  from xml.dom import minidom  
  import xmltodict  
  import sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  interface\_filter = """  
  <filter>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </filter>  
  """  
  loopback = {"int\_name": "Loopback102",  
   "description": "Demo interface by NETCONF",  
   "ip": "192.168.102.1",  
   "netmask": "255.255.255.0"}  
  m = manager.connect(host = device["address"],  
   port = device["netconf\_port"],  
   username = device["username"],  
   password = device["password"],  
   hostkey\_verify = False)

1. Verify NETCONF connection is active

* m.connected

1. Create new config template to delete an interface

* config\_data = """  
  <config>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface operation="delete">  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </config>  
  """

1. Create desired NETCONF config payload and execute to delete the interface

* config = config\_data.format(\*\*loopback)  
  r = m.edit\_config(target = "running", config = config)

1. Print OK status (expected output true)

* print("NETCONF RPC OK: {}".format(r.ok))

1. Create a new NETCONF to check on new loopback interface

* filter = interface\_filter.format(int\_name = "Loopback102")

1. Execute a NETCONF using this filter

* r = m.get\_config("running", filter)

1. Pretty print the raw XML to screen (expected output will not include the loopback interface, as you just deleted it)

* xml\_doc = minidom.parseString(r.xml)  
  print(xml\_doc.toprettyxml(indent = " "))

### End the NETCONF Connection

1. Send a RPC request to disconnect the connection.

* m.close\_session()  
  m.connected

1. End the Python interpreter.

* exit()

## CLI with netmiko

1. From the root of the python\_networking repository, change into the exercise directory.

* cd device\_apis/cli

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

### Retrieve Network Configuration Details with CLI with netmiko\_example1.py

1. Import libraries

* from netmiko import ConnectHandler  
  import re  
  import sys

1. Add parent directory to path to allow importing common vars

* sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device

1. Set device\_type for netmiko

* device["device\_type"] = "cisco\_ios"

1. Create a CLI command template

* show\_interface\_config\_temp = "show running-config interface {}"

1. Open CLI connection to device.
   * *Note: Normally you’d use a with block to open connection to device. This avoids needing to manually m.close\_session() at the end of a script, but for interactive use, this format is chosen.*

* ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create desired CLI command

* command = show\_interface\_config\_temp.format("GigabitEthernet2")

1. Verify the command has been created correctly

* print(command)

1. Send command to device

* interface = ch.send\_command(command)

1. Print the raw command output to the screen

* print(interface)

1. Create regular expression searches to parse the output for desired interface details

* name = re.search(r'interface (.\*)', interface).group(1)  
  description = re.search(r'description (.\*)', interface).group(1)

1. Pull out the ip and mask for the interface

* ip\_info = re.search(r'ip address (.\*) (.\*)', interface)  
  ip = ip\_info.group(1)  
  netmask = ip\_info.group(2)

1. Print the desired info to the screen

* print("The interface {name} has ip address {ip}/{mask}".format(  
   name = name,  
   ip = ip,  
   mask = netmask,  
   )  
   )

### Modify Network Configuration Details with CLI with netmiko\_example2.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from netmiko import ConnectHandler  
  import re, sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  device["device\_type"] = "cisco\_ios"  
  show\_interface\_config\_temp = "show running-config interface {}"  
  ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create Python dictionary with new Loopback Details

* loopback = {"int\_name": "Loopback103",  
   "description": "Demo interface by CLI and netmiko",  
   "ip": "192.168.103.1",  
   "netmask": "255.255.255.0"}

1. Create a CLI configuration

* interface\_config = [  
   "interface {}".format(loopback["int\_name"]),  
   "description {}".format(loopback["description"]),  
   "ip address {} {}".format(loopback["ip"], loopback["netmask"]),  
   "no shut"  
  ]

1. Send configuration to device

* output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to retrieve the new configuration.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)

### Delete Network Configuration Details with CLI with netmiko\_example3.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from netmiko import ConnectHandler  
  import re, sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  device["device\_type"] = "cisco\_ios"  
  show\_interface\_config\_temp = "show running-config interface {}"  
  ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create a new CLI configuration to delete the interface.

* interface\_config = [  
   "no interface {}".format(loopback["int\_name"])  
  ]

1. Send configuration to device

* output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to verify configuration removed.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)
* *Note: attempting to view the configuration of a non-existing interface will generate a CLI error. This output is expected, and one of the reasons APIs like NETCONF or RESTCONF are better suited to programmatic interactions.*

### End the CLI connection to the device

1. Disconnect from the device.

* ch.disconnect()

1. End the Python interpreter.

* exit()

## Other Cool Python Stuff

### Introduction to pyATS

[pyATS](https://developer.cisco.com/site/pyats) is a network testing tool developed by Cisco and made available for free, with significant elements of the underlying code open source.

pyATS offers network developers the ability to profile the network state of hardware, interfaces, protocols, etc… before, during and after changes, to ensure the network is operating as designed, and identify problems before the dreaded phone call. To enable this level of robust testing, pyATS offers a standard way to communicate with network elements and standardize the data returned into native Python objects. This core functionality opens up a lot of flexibility on how pyATS can be used by network developers.

In the following exercises, you will get a brief introduction to pyATS to connect and learn about device details.

### Connect and Interact with a Device

1. From the root of the python\_networking repository, change into the exercise directory.

* cd network\_testing/pyats

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

1. Import in pyATS libraries and tools

* from genie.conf import Genie  
  from ats.topology import loader  
  from genie.abstract import Lookup  
  from genie.libs import ops # noqa

1. Read and process the testbed (inventory) file

* genie\_testbed = Genie.init("./default\_testbed.yaml")

1. Create a pyATS device object from testbed

* vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()
* restconf\_headers = {"Accept": "application/yang-data+json"}  
  restconf\_base = "https://{ip}:{port}/restconf/data"  
  interface\_url = restconf\_base + "/ietf-interfaces:interfaces/interface={int\_name}"

1. Create URL GigE2 Config

* url = interface\_url.format(ip = device["address"],  
   port = device["restconf\_port"],  
   int\_name = "GigabitEthernet2"  
   )

1. Check the complete URL you just composed

* print(url)

1. Send RESTCONF request to core1 for GigE2 Config

* r = requests.get(url,  
   headers = restconf\_headers,  
   auth=(device["username"], device["password"]),  
   verify=False)

1. Print returned data

* print(r.text)

1. If REST call was successful, report interesting details.

* if r.status\_code == 200:  
   # Process JSON data into Python Dictionary and use  
   interface = r.json()["ietf-interfaces:interface"]  
   print("The interface {name} has ip address {ip}/{mask}".format(  
   name = interface["name"],  
   ip = interface["ietf-ip:ipv4"]["address"][0]["ip"],  
   mask = interface["ietf-ip:ipv4"]["address"][0]["netmask"],  
   )  
   )  
  else:  
   print("No interface {} found.".format("GigabitEthernet2"))

### Modify Network Configuration Details with RESTCONF with restconf\_example2.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* import requests, urllib3, sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  urllib3.disable\_warnings(urllib3.exceptions.InsecureRequestWarning)  
  restconf\_headers = {"Accept": "application/yang-data+json"}  
  restconf\_base = "https://{ip}:{port}/restconf/data"  
  interface\_url = restconf\_base + "/ietf-interfaces:interfaces/interface={int\_name}"

1. Add additional Content-Type header.

* restconf\_headers["Content-Type"] = "application/yang-data+json"

1. Create dictionary with details on a new loopback interface.

* loopback = {"name": "Loopback101",  
   "description": "Demo interface by RESTCONF",  
   "ip": "192.168.101.1",  
   "netmask": "255.255.255.0"}

1. Setup data body to create new loopback interface

* data = {  
   "ietf-interfaces:interface": {  
   "name": loopback["name"],  
   "description": loopback["description"],  
   "type": "iana-if-type:softwareLoopback",  
   "enabled": True,  
   "ietf-ip:ipv4": {  
   "address": [  
   {  
   "ip": loopback["ip"],  
   "netmask": loopback["netmask"]  
   }  
   ]  
   }  
   }  
  }

1. Create URL

* url = interface\_url.format(ip = device["address"],  
   port = device["restconf\_port"],  
   int\_name = loopback["name"]  
   )

1. Check the complete URL you just composed

* print(url)

1. Send RESTCONF request to device

* r = requests.put(url,  
   headers = restconf\_headers,  
   auth=(device["username"], device["password"]),  
   json = data,  
   verify=False)

1. Check Status Code (expected 201)

* print("Request Status Code: {}".format(r.status\_code))

1. Query for details on the new interface you just created

* # Create URL and send RESTCONF request to core1 for GigE2 Config  
  url = interface\_url.format(ip = device["address"],  
   port = device["restconf\_port"],  
   int\_name = "Loopback101"  
   )  
  r = requests.get(url,  
   headers = restconf\_headers,  
   auth=(device["username"], device["password"]),  
   verify=False)  
    
  # Print returned data  
  print(r.text)

### Delete Network Configuration Details with RESTCONF with restconf\_example3.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* import requests, urllib3, sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  urllib3.disable\_warnings(urllib3.exceptions.InsecureRequestWarning)  
  restconf\_headers = {"Accept": "application/yang-data+json"}  
  restconf\_base = "https://{ip}:{port}/restconf/data"  
  interface\_url = restconf\_base + "/ietf-interfaces:interfaces/interface={int\_name}"  
  url = interface\_url.format(ip = device["address"],  
   port = device["restconf\_port"],  
   int\_name = "Loopback101"  
   )

1. Send DELETE request to remove the Loopback.

* r = requests.delete(url,  
   headers = restconf\_headers,  
   auth=(device["username"], device["password"]),  
   verify=False)

1. Check Status Code (expected 204)

* print("Request Status Code: {}".format(r.status\_code))

1. Query for details on the new interface (no output expected, as you just deleted it)

* r = requests.get(url,  
   headers = restconf\_headers,  
   auth=(device["username"], device["password"]),  
   verify=False)

1. Check status code (expected 404)

* print(r.status\_code)

## NETCONF with ncclient

1. From the root of the python\_networking repository, change into the exercise directory.

* cd device\_apis/netconf

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

### Retrieve Network Configuration Details with NETCONF with netconf\_example1.py

1. Import libraries

* from ncclient import manager  
  from xml.dom import minidom  
  import xmltodict  
  import sys

1. Add parent directory to path to allow importing common vars

* sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device

1. Create filter template for an interface

* interface\_filter = """  
  <filter>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </filter>  
  """

1. Open NETCONF connection to device
   * *Note: Normally you’d use a with block to open connection to device. This avoids needing to manually m.close\_session() at the end of a script, but for interactive use, this format is chosen.*

* m = manager.connect(host = device["address"],  
   port = device["netconf\_port"],  
   username = device["username"],  
   password = device["password"],  
   hostkey\_verify = False)

1. Verify NETCONF connection is active (expected output true)

* m.connected

1. Create desired NETCONF filter for a particular interface

* filter = interface\_filter.format(int\_name = "GigabitEthernet2")

1. Execute a NETCONF using the filter

* r = m.get\_config("running", filter)

1. Pretty print raw xml to screen

* xml\_doc = minidom.parseString(r.xml)  
  print(xml\_doc.toprettyxml(indent = " "))

1. Process the XML data into Python Dictionary and use

* interface = xmltodict.parse(r.xml)

1. Pretty Print the full Python (Ordered) Dictionary.

* from pprint import pprint  
  pprint(interface)

1. If RPC returned data, print out the interesting pieces.

* if not interface["rpc-reply"]["data"] is None:  
   # Create Python variable for interface details  
   interface = interface["rpc-reply"]["data"]["interfaces"]["interface"]  
    
   print("The interface {name} has ip address {ip}/{mask}".format(  
   name = interface["name"]["#text"],  
   ip = interface["ipv4"]["address"]["ip"],  
   mask = interface["ipv4"]["address"]["netmask"],  
   )  
   )  
  else:  
   print("No interface {} found".format("GigabitEthernet2"))

### Modify Network Configuration Details with NETCONF with netconf\_example2.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from ncclient import manager  
  from xml.dom import minidom  
  import xmltodict  
  import sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  interface\_filter = """  
  <filter>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </filter>  
  """  
  m = manager.connect(host = device["address"],  
   port = device["netconf\_port"],  
   username = device["username"],  
   password = device["password"],  
   hostkey\_verify = False)

1. Verify NETCONF connection is active

* m.connected

1. Create Python dictionary with new Loopback Details

* loopback = {"int\_name": "Loopback102",  
   "description": "Demo interface by NETCONF",  
   "ip": "192.168.102.1",  
   "netmask": "255.255.255.0"}

1. Create NETCONF template for an interface

* config\_data = """  
  <config>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   <description>{description}</description>  
   <type xmlns:ianaift="urn:ietf:params:xml:ns:yang:iana-if-type">  
   ianaift:softwareLoopback  
   </type>  
   <enabled>true</enabled>  
   <ipv4 xmlns="urn:ietf:params:xml:ns:yang:ietf-ip">  
   <address>  
   <ip>{ip}</ip>  
   <netmask>{netmask}</netmask>  
   </address>  
   </ipv4>  
   </interface>  
   </interfaces>  
  </config>  
  """

1. Create desired NETCONF config payload

* config = config\_data.format(\*\*loopback)

1. Send operation

* r = m.edit\_config(target = "running", config = config)

1. Print OK status (expected output true)

* print("NETCONF RPC OK: {}".format(r.ok))

1. Create a new NETCONF to check on new loopback interface

* filter = interface\_filter.format(int\_name = "Loopback102")

1. Execute a NETCONF using this filter

* r = m.get\_config("running", filter)

1. Pretty print the raw XML to screen

* xml\_doc = minidom.parseString(r.xml)  
  print(xml\_doc.toprettyxml(indent = " "))

### Delete Network Configuration Details with NETCONF with netconf\_example3.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from ncclient import manager  
  from xml.dom import minidom  
  import xmltodict  
  import sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  interface\_filter = """  
  <filter>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface>  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </filter>  
  """  
  loopback = {"int\_name": "Loopback102",  
   "description": "Demo interface by NETCONF",  
   "ip": "192.168.102.1",  
   "netmask": "255.255.255.0"}  
  m = manager.connect(host = device["address"],  
   port = device["netconf\_port"],  
   username = device["username"],  
   password = device["password"],  
   hostkey\_verify = False)

1. Verify NETCONF connection is active

* m.connected

1. Create new config template to delete an interface

* config\_data = """  
  <config>  
   <interfaces xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">  
   <interface operation="delete">  
   <name>{int\_name}</name>  
   </interface>  
   </interfaces>  
  </config>  
  """

1. Create desired NETCONF config payload and execute to delete the interface

* config = config\_data.format(\*\*loopback)  
  r = m.edit\_config(target = "running", config = config)

1. Print OK status (expected output true)

* print("NETCONF RPC OK: {}".format(r.ok))

1. Create a new NETCONF to check on new loopback interface

* filter = interface\_filter.format(int\_name = "Loopback102")

1. Execute a NETCONF using this filter

* r = m.get\_config("running", filter)

1. Pretty print the raw XML to screen (expected output will not include the loopback interface, as you just deleted it)

* xml\_doc = minidom.parseString(r.xml)  
  print(xml\_doc.toprettyxml(indent = " "))

### End the NETCONF Connection

1. Send a RPC request to disconnect the connection.

* m.close\_session()  
  m.connected

1. End the Python interpreter.

* exit()

## CLI with netmiko

1. From the root of the python\_networking repository, change into the exercise directory.

* cd device\_apis/cli

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

### Retrieve Network Configuration Details with CLI with netmiko\_example1.py

1. Import libraries

* from netmiko import ConnectHandler  
  import re  
  import sys

1. Add parent directory to path to allow importing common vars

* sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device

1. Set device\_type for netmiko

* device["device\_type"] = "cisco\_ios"

1. Create a CLI command template

* show\_interface\_config\_temp = "show running-config interface {}"

1. Open CLI connection to device.
   * *Note: Normally you’d use a with block to open connection to device. This avoids needing to manually m.close\_session() at the end of a script, but for interactive use, this format is chosen.*

* ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create desired CLI command

* command = show\_interface\_config\_temp.format("GigabitEthernet2")

1. Verify the command has been created correctly

* print(command)

1. Send command to device

* interface = ch.send\_command(command)

1. Print the raw command output to the screen

* print(interface)

1. Create regular expression searches to parse the output for desired interface details

* name = re.search(r'interface (.\*)', interface).group(1)  
  description = re.search(r'description (.\*)', interface).group(1)

1. Pull out the ip and mask for the interface

* ip\_info = re.search(r'ip address (.\*) (.\*)', interface)  
  ip = ip\_info.group(1)  
  netmask = ip\_info.group(2)

1. Print the desired info to the screen

* print("The interface {name} has ip address {ip}/{mask}".format(  
   name = name,  
   ip = ip,  
   mask = netmask,  
   )  
   )

### Modify Network Configuration Details with CLI with netmiko\_example2.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from netmiko import ConnectHandler  
  import re, sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  device["device\_type"] = "cisco\_ios"  
  show\_interface\_config\_temp = "show running-config interface {}"  
  ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create Python dictionary with new Loopback Details

* loopback = {"int\_name": "Loopback103",  
   "description": "Demo interface by CLI and netmiko",  
   "ip": "192.168.103.1",  
   "netmask": "255.255.255.0"}

1. Create a CLI configuration

* interface\_config = [  
   "interface {}".format(loopback["int\_name"]),  
   "description {}".format(loopback["description"]),  
   "ip address {} {}".format(loopback["ip"], loopback["netmask"]),  
   "no shut"  
  ]

1. Send configuration to device

* output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to retrieve the new configuration.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)

### Delete Network Configuration Details with CLI with netmiko\_example3.py

1. Continuing from previous exercise. If starting from new interpreter, execute these steps.

* from netmiko import ConnectHandler  
  import re, sys  
  sys.path.append("..")  
  from device\_info import vagrant\_iosxe as device  
  device["device\_type"] = "cisco\_ios"  
  show\_interface\_config\_temp = "show running-config interface {}"  
  ch = ConnectHandler(ip = device["address"],  
   port = device["ssh\_port"],  
   username = device["username"],  
   password = device["password"],  
   device\_type = device["device\_type"])

1. Create a new CLI configuration to delete the interface.

* interface\_config = [  
   "no interface {}".format(loopback["int\_name"])  
  ]

1. Send configuration to device

* output = ch.send\_config\_set(interface\_config)

1. Print the raw command output to the screen

* print("The following configuration was sent: ")  
  print(output)

1. Create a CLI command to verify configuration removed.

* command = show\_interface\_config\_temp.format("Loopback103")  
  interface = ch.send\_command(command)  
  print(interface)
* *Note: attempting to view the configuration of a non-existing interface will generate a CLI error. This output is expected, and one of the reasons APIs like NETCONF or RESTCONF are better suited to programmatic interactions.*

### End the CLI connection to the device

1. Disconnect from the device.

* ch.disconnect()

1. End the Python interpreter.

* exit()

## Other Cool Python Stuff

### Introduction to pyATS

[pyATS](https://developer.cisco.com/site/pyats) is a network testing tool developed by Cisco and made available for free, with significant elements of the underlying code open source.

pyATS offers network developers the ability to profile the network state of hardware, interfaces, protocols, etc… before, during and after changes, to ensure the network is operating as designed, and identify problems before the dreaded phone call. To enable this level of robust testing, pyATS offers a standard way to communicate with network elements and standardize the data returned into native Python objects. This core functionality opens up a lot of flexibility on how pyATS can be used by network developers.

In the following exercises, you will get a brief introduction to pyATS to connect and learn about device details.

### Connect and Interact with a Device

1. From the root of the python\_networking repository, change into the exercise directory.

* cd network\_testing/pyats

1. Start an interactive Python interpreter. Example below:

* # ipython  
    
  Python 3.6.5 (default, Apr 10 2018, 17:08:37)  
  Type 'copyright', 'credits' or 'license' for more information  
  IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.  
    
  In [1]:

1. Import in pyATS libraries and tools

* from genie.conf import Genie  
  from ats.topology import loader  
  from genie.abstract import Lookup  
  from genie.libs import ops # noqa

1. Read and process the testbed (inventory) file

* genie\_testbed = Genie.init("./default\_testbed.yaml")

1. Create a pyATS device object from testbed

* vagrant\_iosxe1 = genie\_testbed.devices["vagrant-iosxe1"]

1. Connect to the device

* vagrant\_iosxe1.connect()
  + pyATS establishes a connection to the device

1. Create an abstract device to standardize Python API and code for platform

* vagrant\_iosxe1\_abstract = Lookup.from\_device(vagrant\_iosxe1)

1. Using the abstract device, learn about the Interfaces on the end device

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Print out the interface details that were learned

* vagrant\_iosxe1\_interfaces.info

1. Display a single interface from the device

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]

1. Print the mac address for the interface

* vagrant\_iosxe1\_interfaces.info["GigabitEthernet1"]["mac\_address"]

1. Notice that there was no parsing of command line output needed to access this data
2. Execute a command on the device and print the output

* print(vagrant\_iosxe1.execute("show version"))

1. Or store the output into a variable

* version = vagrant\_iosxe1.execute("show version")

1. Send a configuration command to the device

* vagrant\_iosxe1.configure("ntp server 10.10.10.10")

1. Create a configuration command list and send to the device

* config\_loopback = [  
   "interface Loopback201",  
   "description Configured by pyATS",  
   "ip address 172.16.201.1 255.255.255.0",  
   "no shut"  
   ]  
  vagrant\_iosxe1.configure(config\_loopback)

1. Re-learn the interfaces

* vagrant\_iosxe1\_interfaces = vagrant\_iosxe1\_abstract.ops.interface.interface.Interface(vagrant\_iosxe1)  
  vagrant\_iosxe1\_interfaces.learn()

1. Get details about new interface

* vagrant\_iosxe1\_interfaces.info["Loopback201"]

1. Disconnect from the devices

* vagrant\_iosxe1.disconnect()