

# Charming the Python on Nexus 7000 Switch v1

Last Updated: 28-SEP-2015

## About This Solution

The Cisco Nexus 7000 Series devices support Python v2.7.2 in both interactive and non-interactive (script) modes. The Python scripting capability on the Cisco Nexus 7000 Series devices gives programmatic access to the switch command line interface (CLI) to perform various tasks and Power-On Auto Provisioning (POAP) or Embedded Event Manager (EEM) actions.

## About This Demonstration

This demonstration requires familiarity with the Nexus operating system (NxOS), a basic understanding of programming, and motivation to understand Python scripting. The purpose of this demonstration is to showcase different scenarios in which the Nexus 7000 Python API can be used to interact with the Nexus switch.

In this demonstration, you will run through five scenarios that will cover some basic Python programming, and introduce examples on how the Python API can be used to interact with the Nexus switch:

- **Scenario 1.** Python Interpreter
- **Scenario 2.** Basic Python Programming
- **Scenario 3.** Configure interface description using CDP neighbor table
- **Scenario 4.** Configure interface description based on CDP XML elements
- **Scenario 5.** Configure IPv6 addresses and BGPv6 protocol

This demonstration is built using Virtual Internet Routing Lab (VIRL). VIRL is a multi-purpose network virtualization platform that provides an easy way to build, configure, and test new or existing network topologies with an intuitive user interface.

For additional information on the Nexus Python API, go to [N7K Python API](#).

For additional information on VIRL, watch [VIRL \(Virtual Internet Routing Lab\)](#).

## Requirements

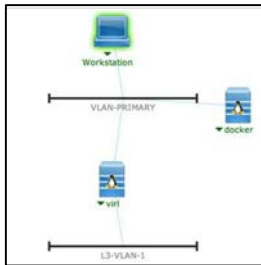
**Table 1.** Requirements

Required	Optional
<ul style="list-style-type: none"> <li>• Laptop</li> </ul>	<ul style="list-style-type: none"> <li>• Cisco AnyConnect</li> <li>• Remote Desktop Client application</li> </ul>

## Topology

This demonstration contains preconfigured users and components to illustrate the capabilities available to you within this sandbox.

You can see the IP address and user account credentials to use to access a component by clicking the component icon in the **Topology** menu of your active session and in the scenario steps that require their use.

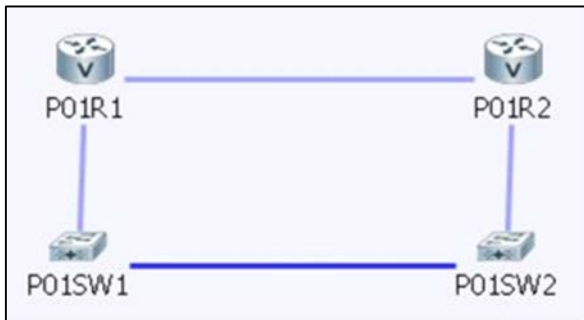
**Figure 1.** Topology

The VIRL topology used for this demonstration, depicted as a single server above, actually consists of:

- Two Nexus switches and two routers interconnected within the topology.

The figure below depicts the VIRL topology.

- All activity will be done on the Nexus P01SW01 switch which has the hostname CiscoLive\_N7K.

**Figure 2.** VIRL Topology

## Get Started

### BEFORE PRESENTING

We strongly recommend that you go through this document and work with an active session before presenting in front of a live audience. This will allow you to become familiar with the structure of the document and content.

### PREPARATION IS KEY TO A SUCCESSFUL PRESENTATION.

Follow the steps to schedule a session and configure your environment.

1. Browse to [dcloud.cisco.com](https://dcloud.cisco.com), select the location closest to you, and log in with your Cisco.com credentials.
2. Schedule a session. [\[Show Me How\]](#)
3. Test your connection. [\[Show Me How\]](#)
4. Verify that the status of your session is **Active** in **My Dashboard > My Sessions**.


**NOTE:** It may take up to 30 minutes for your session to become active.

5. Click **View** to open the active session.

**IMPORTANT:** An Active session does not indicate that the demo is fully launched. It may take up to 10 minutes for this demonstration to fully launch AFTER the simulation is MANUALLY started within VIRL.

6. For best performance, connect to the workstation with **Cisco AnyConnect VPN** [\[Show Me How\]](#) and the **local RDP client on your laptop** [\[Show Me How\]](#)
  - Workstation: **198.18.133.252**, Username: **Administrator**, Password: **C1sco12345**

**NOTE:** You can also connect to the workstation using the Cisco dCloud Remote Desktop client [\[Show Me How\]](#). The dCloud Remote Desktop client works best for accessing an active session with minimal interaction.

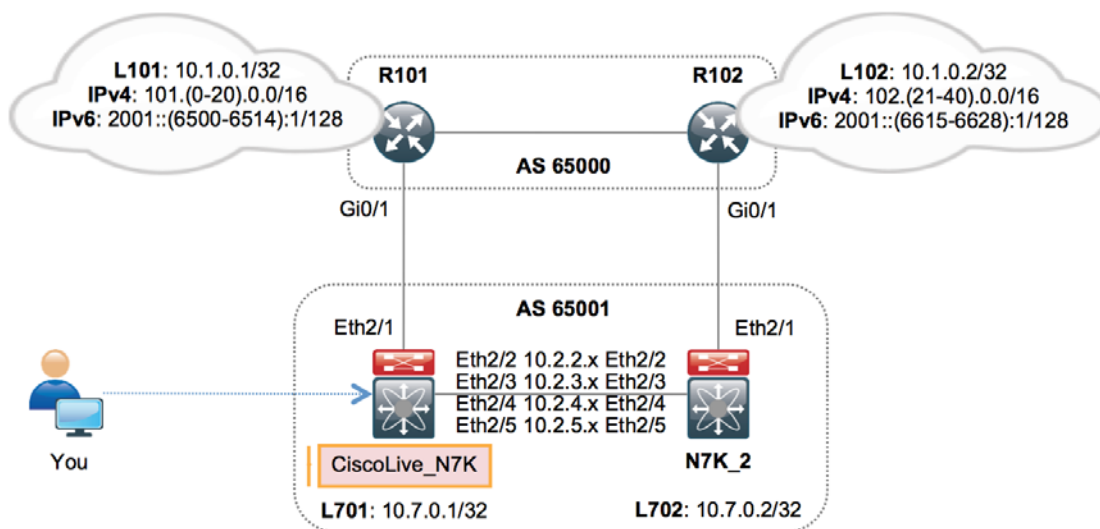
7. From the workstation desktop, double-click the VM Maestro icon () to launch VM Maestro.
  - Once VM Maestro launches, it should have a topology open with the name 'dcloud\_lab\_kt\_pod01-05.virl' under 'My Topologies'.
  - The Nexus switch **P01SW01** will not be available until the demonstration has **MANUALLY** been launched, and completed all launch processes. To MANUALLY launch the demonstration click the green play button.
8. Connect to the Nexus 7000 P01SW01 using **Telnet**.
  - In the 'Simulations' tab four devices will be listed and in the [ACTIVE] state once the simulation has successfully launched.
  - Right click the **P01SW01[ACTIVE]** device, move the mouse cursor over the 'Telnet' menu option, and click the remote access information for the **console port**.
  - This will launch a putty terminal session to the Nexus switch.
  - Alternatively, you can also directly connect to the Nexus switch using a Telnet application from your laptop or desktop computer, using the console IP address **198.18.134.1** with the associated **port\_number**.

- The port number for the console session that you see when you right click on the **P01SW01[ACTIVE]** device.
- All activities will only be done on the Nexus switch P01SW01, which has a node name of **CiscoLive\_N7K**. To login to the Nexus 7000 use the username: **guest** and the password: **guest**. You should be at the prompt: **CiscoLive\_N7K#**

**NOTE:** When using telnet to the Nexus 7000 P01SW01 device omit the colon (e.g. telnet 198.18.134.1 17000). Also omit the colon if entering the port number into your terminal application.

## Overview of the Lab Topology

Figure 3. Lab Topology



Execute these CLI commands from the CiscoLive\_N7K# prompt

```
show hostname
show cdp neighbor
show cdp neighbor | xml
show ip int brief
show ip route local
show ip route ospf
show bgp ipv4 unicast summary
show ip route bgp
```

### IPv6 will be configured in scenario 5

```
show ipv6 int brief
show bgp ipv6 unicast summary
show ipv6 route
```

## Scenario 1. Python Interpreter

### Steps

In this scenario, you will access the Python interpreter on the Nexus switch and use some of the available APIs.

1. Enter 'python' from the enable prompt to access the interpreter.

```
CiscoLive_N7K# python
Copyright (c) 2001-2012 Python Software Foundation; All Rights Reserved
CiscoLive_N7K# >>>
```

**NOTE:** The expected prompt will be in orange, and the output will be in red.

2. Note the prompt change to '>>>'. Enter 'exit' from the interpreter to return to the enable prompt '#'.

```
CiscoLive_N7K# >>> exit
CiscoLive_N7K#
```

**IMPORTANT:** When you exit the interpreter all code is lost. Be aware of the prompt changes during the exercises!!!

### Python Interpreter – Help Function

1. Enter 'help()', this calls the help function. Read the screen output for additional information on the function.

```
CiscoLive_N7K# >>> help()
Welcome to Python 2.7! This is the online help utility.
<Truncated output>
```

2. Enter 'modules'. This will output the available modules.

```
CiscoLive_N7K# help> modules
Please wait a moment while I gather a list of all available modules...
<Truncated output>
CiscoLive_N7K# help>
```

3. Enter 'cisco'. View the contents of the cisco module. We will use the cisco module to access the CLI API.

```
CiscoLive_N7K# help> cisco
Help on module cisco:
NAME
    cisco - commands that integrate with CLI
<Truncated output>
CiscoLive_N7K# help>
```

4. Hit the 'Enter' key twice at the help> prompt to exit help.

```
CiscoLive_N7K# help>
```

```
CiscoLive_N7K# >>>
```

5. If you want to ask for help on a particular object directly from the interpreter, you can enter: 'help("object")'

```
CiscoLive_N7K# >>> help("pprint")
```

```
<Truncated output>
```

6. Omitting the double-quotes will fail. Importing the module is required to not use double-quotes.

```
CiscoLive_N7K# >>> help(pprint)
```

```
Traceback (most recent call last):
```

```
<Truncated output>
```

```
CiscoLive_N7K# >>> import pprint
```

```
CiscoLive_N7K# >>> help(pprint)
```

```
<Truncated output>
```

## Python Interpreter – CLI Command APIs

1. Execute CLI commands from the interpreter using the command 'cli()' API. Notice the output contains control/special characters.

```
CiscoLive_N7K# >>> cli("show hostname")
```

```
'CiscoLive_N7K \n'
```

```
CiscoLive_N7K# >>> cli("show vlan")
```

```
<Truncated output>
```

2. The 'clip()' API outputs directly to stdout and returns nothing to Python, but is more readable.

```
CiscoLive_N7K# >>> clip("show hostname")
```

```
CiscoLive_N7K
```

```
CiscoLive_N7K# >>> clip("show vlan")
```

```
<Truncated output>
```

3. For CLI commands that support XML the 'clid()' API returns outputs as a Python dictionary.

```
CiscoLive_N7K# >>> clid("show hostname")
```

```
{'hostname': 'CiscoLive_N7K'}
```

```
CiscoLive_N7K# >>>
```

4. Assign the 'clid()' output to the variable called hostname. Check the type of the object. Note that 'hostname' is a dict. Extract the value of the key 'hostname' from the variable hostname.

```
CiscoLive_N7K# >>> hostname = clid("show hostname")
```

```
CiscoLive_N7K# >>> type(hostname)
```

```
<type 'dict'>
```

```
CiscoLive_N7K# >>> hostname['hostname']
```

```
'CiscoLive_N7K'
```

5. Change the hostname of the switch. Separate multiple commands with a semi-colon within double quotes. Notice the prompt changed to config-mode.

```
CiscoLive_N7K# >>> cli("conf t ; hostname somethingelse")
```

```
''
```

```
somethingelse(config)#
```

```
somethingelse(config)# >>>
```

6. Change the hostname back to the original name.

```
somethingelse(config)# >>> cli("hostname CiscoLive_N7K")
```

```
''
```

```
CiscoLive_N7K(config)#
```

```
CiscoLive_N7K(config)# >>>
```

7. Exit configuration mode.

```
CiscoLive_N7K(config)# >>> cli("exit")
```

```
''
```

```
CiscoLive_N7K# >>>
```

## Scenario 2. Basic Python Programming

### Steps

In this section, you will use Python variables, use Python script mode, import modules, define functions, and create a for-loop.

#### Basic Python Programming – Variables

1. Concatenate variables var1 and var2 and change var2 to upper case.

```
CiscoLive_N7K# >>> var1 = "charming the python"
CiscoLive_N7K# >>> var2 = " at CiscoLive %s" % '2015'
CiscoLive_N7K# >>> var1 + var2
'charming the python at CiscoLive 2015'
CiscoLive_N7K# >>> var1 + var2.upper()
'charming the python AT CISCOLIVE 2015'
```

#### Basic Python Programming – Python Script

1. Create a Python script file and write it to the default script directory 'bootflash:scripts'. All scripts must be stored in bootflash:scripts or in a sub-directory. From the enable prompt view the contents of the script using the show file ... command. From the enable prompt run the script using the 'source' command.

```
CiscoLive_N7K# >>> import os
CiscoLive_N7K# >>> os.chdir("/bootflash/scripts")
CiscoLive_N7K# >>> f = open("CiscoLive2015.py", 'w')
CiscoLive_N7K# >>> f.write("#!/usr/bin/env python\n")
CiscoLive_N7K# >>> f.write("import sys\n")
CiscoLive_N7K# >>> f.write("args = sys.argv[1:]\n")
CiscoLive_N7K# >>> f.write("for arg in args:\n")
CiscoLive_N7K# >>> f.write("    print 'Welcome to CiscoLive2015 ' + arg\n")
CiscoLive_N7K# >>> f.close()
CiscoLive_N7K# >>> exit
CiscoLive_N7K# show file bootflash:scripts/CiscoLive2015.py
CiscoLive_N7K# source CiscoLive2015.py <your_name>
Welcome to CiscoLive2015 <your_name>
```



## Basic Python Programming – Import Module

1. Import the math module and calculate the area of a circle.

```
CiscoLive_N7K# >>> import math

CiscoLive_N7K# >>> radius = 10

CiscoLive_N7K# >>> circ_area = math.pi * radius ** 2

CiscoLive_N7K# >>> circ_area

314.1592653589793

CiscoLive_N7K# >>>
```

## Basic Python Programming – Python Function

2. Create a function called 'areaCircle' that will calculate the area of a circle when given the argument radius. The interpreter prompt '...' indicates a code block and you must indent with 4 spaces before typing any code, after the function header!!!

**NOTE:** Errors can occur if you do not have the correct indentation.

```
CiscoLive_N7K# >>> def areaCircle(radius):

CiscoLive_N7K# ...     return math.pi * radius ** 2

CiscoLive_N7K# ...

CiscoLive_N7K# >>>

CiscoLive_N7K# >>> areaCircle(10)

CiscoLive_N7K# ...

314.1592653589793

CiscoLive_N7K# >>>
```

## Basic Python Programming – For Loop

1. Create a for-loop that calculates the area of a circle for 3 different random radius. Your integer values will be different.

```
CiscoLive_N7K# >>> from random import randint

CiscoLive_N7K# >>> radius = []

CiscoLive_N7K# >>> for iter in range(3):

CiscoLive_N7K# ...     radius.append(randint(1, 10))

CiscoLive_N7K# ...

CiscoLive_N7K# >>> type(radius), radius

<type 'list'> [8, 6, 8]

CiscoLive_N7K# >>> for integer in radius:
```

```

CiscoLive_N7K# ...      areaCircle(integer)

CiscoLive_N7K# ...

201.06192982974676

113.09733552923255

201.06192982974676

```

## Basic Python Programming – getIPv6 Function

1. From now on the interpreter prompt and enable prompt will be abbreviated. Create a function that converts an IPv4 decimal address to a IPv6 hexadecimal address. NOTE: Remember indentation and watch for typos e.g. spaces, spelling and special characters. Python is an unforgiving language.

```

>>> ipv4_addr = "10.7.0.1"

>>> def getIPv6(ipv4addr):
...     ipv6address = ["2001:"]
...     hexadecimal = []
...     ipv4_addr = ipv4addr.split('.')
...     for item in ipv4_addr:
...         hexadecimal.append(hex(int(item))[2:].zfill(2))
...     for i, j in zip(hexadecimal[::2], hexadecimal[1::2]):
...         ipv6address.append(":" + i + j)
...     return "".join(ipv6address)
...
>>>

>>> print getIPv6(ipv4_addr)

2001::0a07:0001

```

## Basic Python Programming – getIPv6 Module

1. A convertipv4.py script file was created in the bootflash:scripts directory. The convertipv4.py script file can be imported as a module and used in your code. You can call the method 'getIPv6' to convert an IPv4 address.

```

# show file bootflash:scripts/convertipv4.py

>>> from convertipv4 import getIPv6

>>> getIPv6("10.7.0.1")

'2001::0a07:0001'

```

## Scenario 3. Configure interface description using CDP neighbor table

In this section, you will use Python to configure interface descriptions using the information from the CDP table.

### Steps

#### Configure interface description using CDP neighbor table

1. Using the CLI API we will capture the output of the CDP table. We only need the specific neighbor information. So we will start to remove unnecessary lines of data. When for-loop completes it will display which items were popped. The 1st six lines are removed.

```
>>> from cisco import *
>>> import pprint
>>> cdp_dict = dict()
>>> print cdp_dict.items()
[]
>>> cdp_table = cli("show cdp neighbor").strip().split("\n")
>>> print cdp_table
>>> pprint.pprint(cdp_table)
>>> cdp_table[0]
>>> cdp_table[-1]
>>> for iter in range(6):
>>>     cdp_table.pop(0)
...
>>>
```

2. Create `cdp_list1`, using list comprehension, to add each `cdp_table` line as an individual list within `cdp_list1`. Use `split()` to split items within each list. Create `cdp_list2` and only add `cdp_list1` items that are not empty.

```
>>> pprint.pprint(cdp_table)
>>> cdp_list1 = [line.split() for line in cdp_table]
>>> pprint.pprint(cdp_list1)
>>> cdp_list2 = [item for item in cdp_list1 if item != []]
>>> pprint.pprint(cdp_list2)
[['R101 '],
 ['Eth2/1', '144', 'R', 'B', 'IOSv', 'Gig0/1'],
 ['N7K_2(TB3EB2EEBDB)'],
 ['Eth2/2', '136', 'R', 'S', 'S', 'N7K-C7018', 'Eth2/2']]
```

```
['N7K_2(TB3EB2EEBDB)'],
['Eth2/3', '136', 'R', 'S', 's', 'N7K-C7018', 'Eth2/3'],
['N7K_2(TB3EB2EEBDB)'],
['Eth2/4', '136', 'R', 'S', 's', 'N7K-C7018', 'Eth2/4'],
['N7K_2(TB3EB2EEBDB)'],
['Eth2/5', '136', 'R', 'S', 's', 'N7K-C7018', 'Eth2/5'],
['Total', 'entries', 'displayed:', '5']]
```

3. CDP neighbors with large names need to be inserted in the next line and then removed. Also notice there is a list starting with ['Total'...] at the end of the output which must be removed.

```
>>> for item in cdp_list2:
...     item_location = cdp_list2.index(item)
...     if len(item) == 1:
...         cdp_list2[item_location + 1][0:0] = item
...         cdp_list2.pop(item_location)
...     elif 'Total' in item:
...         cdp_list2.pop(item_location)
...
>>>
>>> pprint.pprint(cdp_list2)
[['R101 ', 'Eth2/1', '144', 'R', 'B', 'IOSv', 'Gig0/1'],
 ['N7K_2(TB3EB2EEBDB)', 'Eth2/2', '136', 'R', 'S', 's', 'N7K-C7018', 'Eth2/2'],
 ['N7K_2(TB3EB2EEBDB)', 'Eth2/3', '136', 'R', 'S', 's', 'N7K-C7018', 'Eth2/3'],
 ['N7K_2(TB3EB2EEBDB)', 'Eth2/4', '136', 'R', 'S', 's', 'N7K-C7018', 'Eth2/4'],
 ['N7K_2(TB3EB2EEBDB)', 'Eth2/5', '136', 'R', 'S', 's', 'N7K-C7018', 'Eth2/5']]
```

4. Now add each CDP neighbor into the dictionary `cdp_dict` as a main key of nested dictionary key:values. Add the local/remote interfaces as the nested dictionary key:values. Make the nested dictionary values of type list, to allow for multiple interfaces to be added.

```
>>> for item in cdp_list2:
...     if item[0] not in cdp_dict.keys():
...         cdp_dict[item[0]] = {}
...         cdp_dict[item[0]]['local_intf'] = [item[1]]
...         cdp_dict[item[0]]['remote_intf'] = [item[-1]]
...     elif item[1] not in cdp_dict[item[0]]['local_intf']:
...         cdp_dict[item[0]]['local_intf'].append(item[1])
```

```

...         cdp_dict[item[0]]['remote_intf'].append(item[-1])
...
>>>

>>> print cdp_dict.items()

[('N7K_2(TB3EB2EEBDB)', {'local_intf': ['Eth2/2', 'Eth2/3', 'Eth2/4', 'Eth2/5'],
'remote_intf': ['Eth2/2', 'Eth2/3', 'Eth2/4', 'Eth2/5']}), ('R101 ', {'local_intf':
['Eth2/1'], 'remote_intf': ['Gig0/1']})]

```

5. Now configure a description on all local interfaces, identifying the neighbor and its connected interface. Use the `cdp_dict` key:value pairs to create the description syntax, and configure using the `cli()` API.

```

>>> for key, value in cdp_dict.items():
...     neighbor = ' connected to ' + key + "'s"
...     for item in range(len(value['local_intf'])):
...         local_intf = 'local intf ' + value['local_intf'][item]
...         remote_intf = ' remote intf ' + value['remote_intf'][item]
...         cli("conf t ; interface " + value['local_intf'][item])
...         cli("description " + local_intf + neighbor + remote_intf + " (via screen
scraping)")
...         cli("end")
...
>>>

```

6. Confirm interface description configured is using the CDP information.

```

# show run int eth2/1-5 | in interface|description
!Command: show running-config interface Ethernet2/1-5
interface Ethernet2/1
    description local intf Eth2/1 connected to R101 's remote intf Gig0/1 (via screen scraping)
interface Ethernet2/2
    description local intf Eth2/2 connected to N7K_2(TB3EB2EEBDB)'s remote intf Eth2/2 (via
screen scraping)
interface Ethernet2/3
    description local intf Eth2/3 connected to N7K_2(TB3EB2EEBDB)'s remote intf Eth2/3 (via
screen scraping)
interface Ethernet2/4
    description local intf Eth2/4 connected to N7K_2(TB3EB2EEBDB)'s remote intf Eth2/4 (via
screen scraping)
interface Ethernet2/5

```

```
description local intf Eth2/5 connected to N7K_2(TB3EB2EEBDB)'s remote intf Eth2/5 (via  
screen scraping)
```

## Scenario 4. Configure interface description based on CDP XML elements

In this section, you will use Python to configure interface descriptions using the CDP table XML element tree.

### Steps

#### Configure interface description based on CDP XML elements

1. Review the XML tree data structure and the hierarchy and element names. The XML namespace is assigned to `xmns`.

```
>>> from cisco import *

>>> import xml.etree.cElementTree as etree

>>> cdp_dict = dict()

>>> print cdp_dict.items()

[]

>>> cdp_xml = cli("show cdp neighbor | xml | exclude ']]>]]'")

>>> print cdp_xml

>>> xml_namespace = "{http://www.cisco.com/nxos:1.0:cdpd}"

>>> root = etree.fromstring(cdp_xml)

>>> print root.tag
```

2. Using the XML namespace and an element as a starting point, you can iterate and extract specific data. The CDP neighbor table info is assigned specific XML element names such as `'device_id'`, `'intf_id'` and `'port_id'`.

```
>>> for element in root.iter(xml_namespace + 'ROW_cdp_neighbor_brief_info'):

...     neighbor = element.find(xml_namespace + 'device_id').text

...     local_intf = element.find(xml_namespace + 'intf_id').text

...     remote_intf = element.find(xml_namespace + 'port_id').text

...     if neighbor not in cdp_dict:

...         cdp_dict[neighbor] = {}

...         cdp_dict[neighbor]['local_intf'] = [local_intf]

...         cdp_dict[neighbor]['remote_intf'] = [remote_intf]

...     elif local_intf not in cdp_dict[neighbor]['local_intf']:

...         cdp_dict[neighbor]['local_intf'].append(local_intf)

...         cdp_dict[neighbor]['remote_intf'].append(local_intf)

...

>>>
```

3. Assign the neighbor and associated local and remote interfaces into a nested dictionary.

```
>>> print cdp_dict.items()

[('N7K_2(TB3EB2EEBDB)', {'local_intf': ['Ethernet2/2', 'Ethernet2/3', 'Ethernet2/4', 'Ethernet2/5'],
'remote_intf': ['Ethernet2/2', 'Ethernet2/3', 'Ethernet2/4', 'Ethernet2/5']}),
('R101 ', {'local_intf': ['Ethernet2/1'], 'remote_intf': ['GigabitEthernet0/1']})]
```

4. Now configure a description on all local interfaces using XML data.

```
>>> for key, value in cdp_dict.items():
...     neighbor = " connected to " + key + "'s"
...     for item in range(len(value['local_intf'])):
...         local_intf = " local intf " + value['local_intf'][item]
...         remote_intf = " remote intf " + value['remote_intf'][item]
...         cli("conf t ; interface " + value['local_intf'][item])
...         cli("description " + local_intf + neighbor + remote_intf + " (via XML)")
...         cli("end")
...
>>>
```

5. Confirm interface description configured is using the CDP information.

```
# show run int eth2/1-5 | in interface|description
!Command: show running-config interface Ethernet2/1-5
interface Ethernet2/1
    description local intf Ethernet2/1 connected to R101 's remote intf GigabitEthernet0/1 (via XML)
interface Ethernet2/2
    description local intf Ethernet2/2 connected to N7K_2(TB3EB2EEBDB)'s remote intf Ethernet2/2 (via XML)
interface Ethernet2/3
    description local intf Ethernet2/3 connected to N7K_2(TB3EB2EEBDB)'s remote intf Ethernet2/3 (via XML)
interface Ethernet2/4
    description local intf Ethernet2/4 connected to N7K_2(TB3EB2EEBDB)'s remote intf Ethernet2/4 (via XML)
interface Ethernet2/5
    description local intf Ethernet2/5 connected to N7K_2(TB3EB2EEBDB)'s remote intf Ethernet2/5 (via XML)
```



## Scenario 5. Configure IPv6 addresses and BGPv6 protocol

In this section, you will use Python to generate IPv6 addresses by converting existing IPv4 decimal addresses into hexadecimal, and then configure IPv6 addresses on the interfaces. You will also deploy a BGPv6 configlet on the Nexus node using Python to complete the IPv6 routing requirement.

### Steps

#### Configure IPv6 addresses and BGPv6 protocol

1. Import the `convertipv4` module to use the `getIPv6` method. Review the XML tree data structure and the hierarchy and element names. The XML namespace is assigned to `xmlns`.

```
>>> from cisco import *
>>> from convertipv4 import getIPv6
>>> import xml.etree.cElementTree as etree
>>> intf_brief_dict = dict()
>>> print intf_brief_dict.items()
[]
>>> intf_brief_xml = cli("show ip int brief | xml | exclude ']]>]]'")
>>> print intf_brief_xml
>>> xml_namespace = "{http://www.cisco.com/nxos:1.0:ip}"
>>> root = etree.fromstring(intf_brief_xml)
>>> print root.tag
```

2. Using the XML namespace and an element as a starting point, you can iterate and extract specific data. Use the `getIPv6` method to convert the IPv4 address.

```
>>> for element in root.iter(xml_namespace + 'ROW_intf'):
...     prefix = element.find(xml_namespace + 'prefix').text
...     intf_name = element.find(xml_namespace + 'intf-name').text
...     if intf_name not in intf_brief_dict:
...         intf_brief_dict[intf_name] = {}
...         intf_brief_dict[intf_name]['IPv4'] = prefix
...         intf_brief_dict[intf_name]['IPv6'] = getIPv6(prefix)
...
>>> print sorted(intf_brief_dict.items())

[('Eth2/1', {'IPv4': '192.168.101.1', 'IPv6': '2001::c0a8:6501'}), ('Eth2/2', {'IPv4':
'10.2.2.1', 'IPv6': '2001::0a02:0201'}), ('Eth2/3', {'IPv4': '10.2.3.1', 'IPv6':
```

```
'2001::0a02:0301'})), ('Eth2/4', {'IPv4': '10.2.4.1', 'IPv6': '2001::0a02:0401'})), ('Eth2/5',
{'IPv4': '10.2.5.1', 'IPv6': '2001::0a02:0501'})), ('Lo701', {'IPv4': '10.7.0.1', 'IPv6':
'2001::0a07:0001'}))]
```

- Now configure IPv6 address on all IPv4 interfaces, using the cli() API. The loopback701 and Eth2/1-5 interfaces should all have IPv6 addresses configured. The IPv6 routing table now has directly attached routes, and is learning the Loopback702 ipv6 address on N7K\_2 via ospfv3.

```
>>> for key, value in intf_brief_dict.items():
...     intf = key
...     ipv6_addr = value['IPv6']
...     if 'Lo' in intf:
...         ipv6_mask = '/128'
...     else:
...         ipv6_mask = '/126'
...     cli("conf t ; interface " + key)
...     cli("ipv6 address " + ipv6_addr + ipv6_mask)
...     cli("end")
...
>>> clip("show ipv6 int brief")
>>> clip("show ipv6 route")
```

- A bgpv6\_configlet.txt file that contains the required BGPv6 configuration was created in the bootflash:scripts directory. Open and read the file and put into a list.

```
# show file bootflash:scripts/bgpv6_configlet.txt
>>> bgpv6_file = open("/bootflash/scripts/bgpv6_configlet.txt", 'r')
>>> bgpv6_config = bgpv6_file.read().split("\n")
>>> print bgpv6_config
>>> print bgpv6_config[0]
>>> type(bgpv6_config)
```

- Change the interpreter prompt into configuration mode. Iterate over each line in the list and configure the BGPv6 protocol. Confirm BGPv6 peering is up and you are receiving ipv6 routes.

```
>>> cli("conf t")
(config) # >>> for item in bgpv6_config:
(config) # ...     print item
(config) # ...
(config) # >>> for item in bgpv6_config:
(config) # ...     cli(item)
```

```
(config) # ...  
(config-router-neighbor-af) # >>> cli("end")  
  
>>>  
  
>>> clip("show run bgp")  
  
>>> clip("show bgp ipv6 unicast summary")  
  
>>> clip("show ipv6 route")
```



**Americas Headquarters**  
Cisco Systems, Inc.  
San Jose, CA

**Asia Pacific Headquarters**  
Cisco Systems (USA) Pte. Ltd.  
Singapore

**Europe Headquarters**  
Cisco Systems International BV Amsterdam,  
The Netherlands

Cisco has more than 200 offices worldwide. Addresses, phone numbers, and fax numbers are listed on the Cisco Website at [www.cisco.com/go/offices](http://www.cisco.com/go/offices).

Cisco and the Cisco logo are trademarks or registered trademarks of Cisco and/or its affiliates in the U.S. and other countries. To view a list of Cisco trademarks, go to this URL: [www.cisco.com/go/trademarks](http://www.cisco.com/go/trademarks). Third party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (1110R)