1. Sequences

1.1 Strings

Example 1: Common operations with strings and string methods

```
# Example of the string

switch = 'ukdcswa01'
```

```
print switch
```

```
ukdcswa01
```

```
# Using method available to strings, i.e. 'upper' changes all characters to upper case. Original
# variable stays the same, unless result is re-assigned back to the same variable name

print switch.upper()
```

```
UKDCSWA01
```

```
# Slicing string, i.e. taking characters 0,1,2,3 and assigning result to a new variable

dc_prefix = switch[0:4]

print dc_prefix
```

```
ukdc
```

```
# Another slicing example, but taking last 2 characters: -2,-1 of the string

switch_number = switch[-2:]

print switch_number
```

```
01
```

```
# Another slicing example: taking characters from index 4, ignoring last two

switch_type = switch[4:-2]

print switch_type
```

```
swa
```

```
# Generate new switch name by concatinating above strings and incrementing switch_number

switch2 = '{0}{1}{2:02d}'.format(dc_prefix, switch_type, int(switch_number) + 1)

# {:02d}.format pads with 0 numbers less than 10 (i.e. 02 for 2)

print switch2
```

```
ukdcswa02
```

```
# Replace Contry code

print switch.replace('uk', 'fr')
```

```
frdcswa01
```

Example 2: Parsing text outputs

```
# Example of output of the interface status command from a switch

output = '''
Port       Name                               Status       Vlan      Duplex Speed  Type          Flags
Et1        SERVER 1                           connected    200       full   10G    10GBASE-CR
Et2        SERVER 2                           connected    200       full   10G    10GBASE-CR
Et3        SERVER 3                           connected    200       full   10G    10GBASE-CR
Et4        SERVER 4                           connected    201       full   10G    10GBASE-CR
Et5        SERVER 5                           connected    202       full   10G    10GBASE-CR
Et6        UNUSED                             disabled     666       full   10G    Not Present
Et7        UNUSED                             disabled     666       full   10G    Not Present
Et8        UNUSED                             disabled     666       full   10G    Not Present
Et9        UNUSED                             disabled     666       full   10G    Not Present
Et10       UNUSED                             disabled     666       full   10G    Not Present
'''

# Converting to the list of lines by splitting by end of line character '\n' and re-using same
# variable name

output = output.split('\n')

# Iterating through items of the list

for line in output:
    print line
```

```
Port       Name                               Status       Vlan      Duplex Speed  Type          Flags
Et1        SERVER 1                           connected    200       full   10G    10GBASE-CR
Et2        SERVER 2                           connected    200       full   10G    10GBASE-CR
Et3        SERVER 3                           connected    200       full   10G    10GBASE-CR
Et4        SERVER 4                           connected    201       full   10G    10GBASE-CR
Et5        SERVER 5                           connected    202       full   10G    10GBASE-CR
Et6        UNUSED                             disabled     666       full   10G    Not Present
Et7        UNUSED                             disabled     666       full   10G    Not Present
Et8        UNUSED                             disabled     666       full   10G    Not Present
Et9        UNUSED                             disabled     666       full   10G    Not Present
Et10       UNUSED                             disabled     666       full   10G    Not Present
```

```
# As each line is an item in a list, we can directly access a line by index, i.e.

print output[2]
```

```
Et1        SERVER 1                           connected    200       full   10G    10GBASE-CR
```

```
print output[3]
```

```
Et2        SERVER 2                           connected    200       full   10G    10GBASE-CR
```

```
# Parsing content of a 4th line (index = 3): split line by empty space into list and use index to
# select item of the list, i.e. first one (index = 0)

print output[3].split()[0]
```

```
Et2
```

```
# Splitting by empty space problematic as fields like Description above have spaces, thus 'SERVER' and '2'
# would appear as separate items in the list, skewing index. As result Description is stored in two items
# of the list:
print output[3].split()
```

```
['Et2', 'SERVER', '2', 'connected', '200', 'full', '10G', '10GBASE-CR']
```

```
# For this reason parsing output lines may need a different approach - fixed-width columns parsing,
# which is commonly applied to screen scraping of text outputs of commands on Cisco, Arista network devices.
# Simple approach is to identify position (index) of Column Name in the header of the table and cut respective
# characters from target line from that index till index of next Column

header = 'Port       Name                               Status       Vlan      Duplex Speed  Type         Flags'

# As example, to identify position of 'Status' in a line:

status_start_idx = header.index('Status')
status_end_idx = header.index('Vlan')

print 'Start index for "Status" column:', status_start_idx

print 'End index for "Status" column:', status_end_idx
```

```
Start index for "Status" column: 47
End index for "Status" column: 60
```

```
# Position of 'Port' starts with index = 0 as it is beginning of the line and ends where 'Name' starts

port_start_idx = header.index('Port')
port_end_idx = header.index('Name')

print 'Start index for "Port" column:', port_start_idx

print 'End index for "Port" column:', port_end_idx
```

```
Start index for "Port" column: 0
End index for "Port" column: 11
```

```
# Knowing Columns start and end indexes we can iterate via output list (line by line) and pick Port and Status

for line in output:
    print line[port_start_idx:port_end_idx].strip(), line[status_start_idx:status_end_idx].strip()
```

```
Port Status
Et1 connected
Et2 connected
Et3 connected
Et4 connected
Et5 connected
Et6 disabled
Et7 disabled
Et8 disabled
Et9 disabled
Et10 disabled
```

1.2 Lists

```
# Example of list of strings (can be any types, i.e. can be list of dictionaries, list of lists, can be mixed types)

switches = ['ukdcswa01', 'ukdcswa02', 'ukdcswa03']

# Extracting items from the list by index:

print switches[0] # First one
```

```
ukdcswa01
```

```
print switches[-1] # Last one
```

```
ukdcswa03
```

```
# Slicing can be also performed by specifying start and end indexes (end index not included)

print switches[0:2]
```

```
['ukdcswa01', 'ukdcswa02']
```

```
# Append to the list

switches.append('ukdcswa05')

print switches
```

```
['ukdcswa01', 'ukdcswa02', 'ukdcswa03', 'ukdcswa05']
```

```
# Insert into the list, i.e. into index 3

switches.insert(3, 'ukdcswa04')

print switches
```

```
['ukdcswa01', 'ukdcswa02', 'ukdcswa03', 'ukdcswa04', 'ukdcswa05']
```

```
# Amend item of the list, i.e. item with index 3

switches[3] = 'ukdcswa06'

print switches
```

```
['ukdcswa01', 'ukdcswa02', 'ukdcswa03', 'ukdcswa06', 'ukdcswa05']
```

```
# Iterating through lists

for s in switches:
    print s
```

```
ukdcswa01
ukdcswa02
ukdcswa03
ukdcswa06
ukdcswa05
```

2. Mapping Data Type (Dictionary)

```
# Dictionary with key = switch_name, value = ip_address

switches = {'ukdcswa01': '1.1.1.1', 'ukdcswa02': '2.2.2.2', 'ukdcswa03': '3.3.3.3'}

print switches
```

```
{'ukdcswa03': '3.3.3.3', 'ukdcswa02': '2.2.2.2', 'ukdcswa01': '1.1.1.1'}
```

```
# Lookup key/value pair by key

print switches['ukdcswa02']
```

```
2.2.2.2
```

```
# Print keys of a dictionary
print switches.keys()
```

```
['ukdcswa03', 'ukdcswa02', 'ukdcswa01']
```

```
# Print values of a dictionary

print switches.values()
```

```
['3.3.3.3', '2.2.2.2', '1.1.1.1']
```

```
# Add new key/value pair into dictionary

switches['ukdcswa04'] = '4.4.4.4'

print switches
```

```
{'ukdcswa03': '3.3.3.3', 'ukdcswa02': '2.2.2.2', 'ukdcswa01': '1.1.1.1', 'ukdcswa04': '4.4.4.4'}
```

```
# Update existing key/value pair
switches['ukdcswa04'] = '4.4.5.4'

print switches
```

```
{'ukdcswa03': '3.3.3.3', 'ukdcswa02': '2.2.2.2', 'ukdcswa01': '1.1.1.1', 'ukdcswa04': '4.4.5.4'}
```

```
# Remove key/value pair

switches.pop('ukdcswa04')

print switches
```

```
{'ukdcswa03': '3.3.3.3', 'ukdcswa02': '2.2.2.2', 'ukdcswa01': '1.1.1.1'}
```

```
# Iterate through dictionary is via keys

for k in switches:
    print k
```

```
ukdcswa03
ukdcswa02
ukdcswa01
```

```
# While iterating via keys extract corresponding value by specifying the key

for k in switches:
    print k, switches[k] # Printing key and extracting value from switches dictionary for that key
```

```
ukdcswa03 3.3.3.3
ukdcswa02 2.2.2.2
ukdcswa01 1.1.1.1
```

```
# 'Items' method of dictionary type creates list of tuples where first item is key and second item is
# value as result in a for loop a key can be assigned to one variable and value to another.

for k,v in switches.items():
    print k,v
```

```
ukdcswa03 3.3.3.3
ukdcswa02 2.2.2.2
ukdcswa01 1.1.1.1
```

```
# Value of a key in above example was a string, but can be list, dictionary, or a combination of those.

switches = {'ukdcswa01': {'ip': '1.1.1.1', 'os': 'cisco_ios'},
            'ukdcswa02': {'ip': '2.2.2.2', 'os': 'nexus_os'},
            'ukdcswa03': {'ip': '3.3.3.3'}}

print switches
```

```
{'ukdcswa03': {'ip': '3.3.3.3'}, 'ukdcswa02': {'ip': '2.2.2.2', 'os': 'nexus_os'}, 'ukdcswa01': {'ip':
'1.1.1.1', 'os': 'cisco_ios'}}
```

```
# Lookup value by key

print switches['ukdcswa01']
```

```
{'ip': '1.1.1.1', 'os': 'cisco_ios'}
```

```
# As value is dictionary itself, to lookup inner dictionary

print switches['ukdcswa01']['ip']
```

```
1.1.1.1
```

```
# To update inner dictionary by adding another key/value pair

switches['ukdcswa03'].update({'os': 'arista_eos'})

print switches
```

```
{'ukdcswa03': {'ip': '3.3.3.3', 'os': 'arista_eos'}, 'ukdcswa02': {'ip': '2.2.2.2', 'os': 'nexus_os'},
'ukdcswa01': {'ip': '1.1.1.1', 'os': 'cisco_ios'}}
```

3. Example parsing JSON

```
output = '''
{
    "modelName": "vEOS",
    "internalVersion": "4.18.5M-6710299.4185M",
    "systemMacAddress": "2c:c2:60:a2:21:e0",
    "serialNumber": "",
    "memTotal": 3887524,
    "bootupTimestamp": 1523305176.32,
    "memFree": 2495920,
    "version": "4.18.5M",
    "architecture": "i386",
    "isIntlVersion": false,
    "internalBuildId": "3d3db99a-d291-4527-86d2-b296f4998d48",
    "hardwareRevision": ""
}

'''
```

```
# Import json module

import json

from pprint import pprint # Pprint is pretty-print and very handy for print data structures like dictionaries
```

```
# Use json module to parse json from string and convert into dictionary

data = json.loads(output)
```

```
# Type of data is Ordered Dictionary (dictionary, where order of key/value pairs is preserved)

print type(data)
```

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

```
pprint(data)
```

```
{u'architecture': u'i386',
 u'bootupTimestamp': 1523305176.32,
 u'hardwareRevision': u'',
 u'internalBuildId': u'3d3db99a-d291-4527-86d2-b296f4998d48',
 u'internalVersion': u'4.18.5M-6710299.4185M',
 u'isIntlVersion': False,
 u'memFree': 2495920,
 u'memTotal': 3887524,
 u'modelName': u'vEOS',
 u'serialNumber': u'',
 u'systemMacAddress': u'2c:c2:60:a2:21:e0',
 u'version': u'4.18.5M'}
```

```
# Lookup values by keys

print data['version']
```

```
4.18.5M
```

```
pprint(data.keys())
```

```
[u'memTotal',
 u'version',
 u'internalVersion',
 u'serialNumber',
 u'systemMacAddress',
 u'bootupTimestamp',
 u'memFree',
 u'modelName',
 u'architecture',
 u'isIntlVersion',
 u'internalBuildId',
 u'hardwareRevision']
```

4. Example parsing XML

```
# Example of output of 'show cdp neighbors | xml' on Nexus switch

output = '''<?xml version="1.0" encoding="ISO-8859-1"?>
<nf:rpc-reply xmlns:nf="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns="http://www.cisco.com/nxos:1.0:cdpd">
 <nf:data>
  <show>
   <cdp>
    <neighbors>
     <__XML__OPT_Cmd_show_cdp_neighbors_interface>
      <__XML__OPT_Cmd_show_cdp_neighbors___readonly__>
       <__readonly__>
        <TABLE_cdp_neighbor_brief_info>
         <ROW_cdp_neighbor_brief_info>
          <ifindex>436301824</ifindex>
          <device_id>ukdcswa02</device_id>
          <intf_id>Ethernet1/40</intf_id>
          <ttl>153</ttl>
          <capability>router</capability>
          <capability>switch</capability>
          <capability>IGMP_cnd_filtering</capability>
          <capability>Supports-STP-Dispute</capability>
          <platform_id>N5K-C5548UP</platform_id>
          <port_id>Ethernet1/40</port_id>
         </ROW_cdp_neighbor_brief_info>
         <ROW_cdp_neighbor_brief_info>
          <ifindex>436305920</ifindex>
          <device_id>ukdcswa02</device_id>
          <intf_id>Ethernet1/41</intf_id>
          <ttl>173</ttl>
          <capability>router</capability>
          <capability>switch</capability>
          <capability>IGMP_cnd_filtering</capability>
          <capability>Supports-STP-Dispute</capability>
          <platform_id>N5K-C5548UP</platform_id>
          <port_id>Ethernet1/41</port_id>
         </ROW_cdp_neighbor_brief_info>
         <ROW_cdp_neighbor_brief_info>
          <ifindex>436310016</ifindex>
          <device_id>ukdcswd01</device_id>
          <intf_id>Ethernet1/42</intf_id>
          <ttl>170</ttl>
          <capability>router</capability>
          <capability>switch</capability>
          <capability>IGMP_cnd_filtering</capability>
          <capability>Supports-STP-Dispute</capability>
          <platform_id>N7K-C7010</platform_id>
          <port_id>Ethernet1/10</port_id>
         </ROW_cdp_neighbor_brief_info>
         <ROW_cdp_neighbor_brief_info>
          <ifindex>436310016</ifindex>
          <device_id>ukdcswd02</device_id>
          <intf_id>Ethernet1/43</intf_id>
          <ttl>170</ttl>
          <capability>router</capability>
          <capability>switch</capability>
          <capability>IGMP_cnd_filtering</capability>
          <capability>Supports-STP-Dispute</capability>
          <platform_id>N7K-C7010</platform_id>
          <port_id>Ethernet1/10</port_id>
         </ROW_cdp_neighbor_brief_info>
        </TABLE_cdp_neighbor_brief_info>
       </__readonly__>
      </__XML__OPT_Cmd_show_cdp_neighbors___readonly__>
     </__XML__OPT_Cmd_show_cdp_neighbors_interface>
    </neighbors>
   </cdp>
  </show>
 </nf:data>
</nf:rpc-reply>
'''
```

```
# Importing XML module and Pretty-print to print formatted data structure

import xmltodict

from pprint import pprint
```

```
data = xmltodict.parse(output)
```

```
# Result of parsing XML (same is json) is ordered dictionary

print type(data)
```

```
<class 'collections.OrderedDict'>
```

```
# Next step is to go layer by layer to get to the inner data

print data.keys() # Get top level keys
```

```
[u'nf:rpc-reply']
```

```
print data['nf:rpc-reply'].keys() # Next level keys
```

```
[u'@xmlns:nf', u'@xmlns', u'nf:data']
```

```
# 'nf:data' is of interest as it contains next level data

print data['nf:rpc-reply']['nf:data'].keys() # Next level keys
```

```
[u'show']
```

```
print data['nf:rpc-reply']['nf:data']['show'].keys() # Next level keys
```

```
[u'cdp']
```

```
print data['nf:rpc-reply']['nf:data']['show']['cdp'].keys() # Next level keys
```

```
[u'neighbors']
```

```
print data['nf:rpc-reply']['nf:data']['show']['cdp']['neighbors'].keys() # Next level keys
```

```
[u'__XML__OPT_Cmd_show_cdp_neighbors_interface']
```

```
print data['nf:rpc-reply']['nf:data']['show']['cdp']['neighbors']\
['__XML__OPT_Cmd_show_cdp_neighbors_interface'].keys()
```

```
[u'__XML__OPT_Cmd_show_cdp_neighbors___readonly__']
```

```
# Notice '\' to continue line on the next one as line becomes too long

print data['nf:rpc-reply']['nf:data']['show']['cdp']['neighbors']\
['__XML__OPT_Cmd_show_cdp_neighbors_interface']\
[u'__XML__OPT_Cmd_show_cdp_neighbors___readonly__'].keys()
```

```
[u'__readonly__']
```

```
print data['nf:rpc-reply']['nf:data']['show']['cdp']['neighbors']\
['__XML__OPT_Cmd_show_cdp_neighbors_interface']\
[u'__XML__OPT_Cmd_show_cdp_neighbors___readonly__']['__readonly__'].keys()
```

```
[u'TABLE_cdp_neighbor_brief_info']
```

```
print data['nf:rpc-reply']['nf:data']['show']['cdp']['neighbors']\
['__XML__OPT_Cmd_show_cdp_neighbors_interface']\
[u'__XML__OPT_Cmd_show_cdp_neighbors___readonly__']['__readonly__']\
[u'TABLE_cdp_neighbor_brief_info'].keys()
```

```
[u'ROW_cdp_neighbor_brief_info']
```

```
# This last key contains the inner data, which is a list for dictionaries. Re-assign that to a new variable
cdp_neighbors = data['nf:rpc-reply']['nf:data']['show']['cdp']['neighbors']\
['__XML__OPT_Cmd_show_cdp_neighbors_interface']\
[u'__XML__OPT_Cmd_show_cdp_neighbors___readonly__']\
['__readonly__'][u'TABLE_cdp_neighbor_brief_info']\
[u'ROW_cdp_neighbor_brief_info']

print type(cdp_neighbors)
```

```
<type 'list'>
```

```
# Use index to get item out of the list, i.e. second one with Index 1 (as indexes start from 0).
print cdp_neighbors[1]
```

```
OrderedDict([(u'ifindex', u'436305920'), (u'device_id', u'ukdcswa02'), (u'intf_id', u'Ethernet1/41'),
(u'ttl', u'173'), (u'capability', [u'router', u'switch', u'IGMP_cnd_filtering', u'Supports-STP-Disput
e']), (u'platform_id', u'N5K-C5548UP'), (u'port_id', u'Ethernet1/41')])
```

```
# Ordered Dict is a dictionary with preserved order of key/value pairs, presented as tuple,
# where first item in a tuple is a key and second one is a value.
print cdp_neighbors[1]['device_id']
```

```
ukdcswa02
```