Python Network Libraries

Module Overview

- Python Libraries
- argparse
- Regular Expressions 101
- TextFSM
- NAPALM

Command Line Arguments

Introduction to Python for Network Engineers

User interaction - Prompting users for input

- The `raw_input` (Python 2.x) or `input` (Python 3.x) built-in function is used to collect user input, interactively
- Prompt the user for data that can then be stored as a variable and used in the script

```
number_of_routers = input('Enter the number of routers in the mesh:')

num_routers = int(number_of_routers)

number_of_connections = ( num_routers * (num_routers - 1) )/2

print("For a full mesh of {} routers, you will need {} connections".format(num_routers, number_of_connections))
```

Passing in Arguments

- Using the `sys` module
  - `argv` is a attribute (variable) within the `sys` module that makes it fast and easy to pass variables in from the command line
- Using `argparse` module
  - Built-in module that allows for more functionality such as defining a help menu and using user-friendly flags

sys.argv

- It's a variable that is of type `list`

```
#! /usr/bin/env python

import sys

if __name__ == "__main__":

  print(sys.argv)
```

sys.argv

- It's a variable that is of type `list`

```
#! /usr/bin/env python

import sys

if __name__ == "__main__":

  print(sys.argv)
```

- Script name is `argv[0]`

```
ntc@jump-host:~$ python args_test.py hello world 10.1.1.1 NYCR1
['args_test.py', 'hello', 'world', '10.1.1.1', 'NYCR1']
```

Example - sys.argv

Objective:

- Pass in the "fact" you want to see the value for and the proper key-value pair will be printed from the `facts` dictionary.

Dictionary:

```
facts = {'vendor': 'cisco', 'mgmt_ip': '10.1.1.1', 'model': 'nexus', 'hostname': 'NYC301', 'os': '6.1.2'}
```

User experience:

```
ntc@jump-host:~$ python print_facts.py model
model: nexus
```

Examining the Code

```python
#! /usr/bin/env python

import sys

if __name__ == "__main__":

    facts = {'vendor': 'cisco', 'mgmt_ip': '10.1.1.1', 'model': 'nexus', 'hostname': 'NYC301', 'os': '6.1.2'}

    args = sys.argv        # assign argv to args (optional; cleans up the code)

    fact_to_print = args[1]      # assign the second element to my_fact

    print(fact_to_print + ': ' + facts[fact_to_print])

    print(args)         # added for example below
```

```
ntc@jump-host:~$ python print_facts.py model
model: nexus
['print_facts.py', 'model']
```

argparse

- Python module that simplifies defining a help menu, using user-friendly flags, and much more

```
ntc@jump-host:~$ python get_facts.py -f model
model: nexus
ntc@jump-host:~$ python get_facts.py -f=model
model: nexus
ntc@jump-host:~$ python get_facts.py --f=model
model: nexus
ntc@jump-host:~$ python get_facts.py --fact=model
model: nexus
ntc@jump-host:~$ python get_facts.py --fact model
model: nexus
```

argparse

- Python module that simplifies defining a help menu, using user-friendly flags, and much more

```
ntc@jump-host:~$ python get_facts.py -f model
model: nexus
ntc@jump-host:~$ python get_facts.py -f=model
model: nexus
ntc@jump-host:~$ python get_facts.py --f=model
model: nexus
ntc@jump-host:~$ python get_facts.py --fact=model
model: nexus
ntc@jump-host:~$ python get_facts.py --fact model
model: nexus
```

```python
import argparse

if __name__ == "__main__":

    facts = {'vendor': 'cisco', 'mgmt_ip': '10.1.1.1', 'model': 'nexus', 'hostname': 'NYC301', 'os': '6.1.2'}

    parser = argparse.ArgumentParser(description='Python Argparse Demo')
    parser.add_argument('-f', '--fact', help='enter a valid fact from the device facts dictionary')

    args = parser.parse_args()

    print(args.fact + ': ' + facts[args.fact])
```

argparse - Built-in help

- Leverage help menu natively built-in
- Can be disabled if needed when parser is instantiated

```
ntc@jump-host:~$ python get_facts.py --help
usage: get_facts.py [-h] [-f FACT]

Python Argparse demo.

optional arguments:
  -h, --help            show this help message and exit
  -f FACT, --fact FACT  enter a valid fact from the device facts dictionary
```

argparse - Choices

- Built-in error validation
- What if the user enters an invalid value for argument?

```
ntc@jump-host:~$ python get_facts.py --f platform
Traceback (most recent call last):
  File "get_facts.py", line 14, in <module>
    print(args.fact + ': ' + facts[args.fact])
KeyError: 'platform'
```

argparse - Choices

- Built-in error validation
- What if the user enters an invalid value for argument?

```
ntc@jump-host:~$ python get_facts.py --f platform
Traceback (most recent call last):
  File "get_facts.py", line 14, in <module>
    print(args.fact + ': ' + facts[args.fact])
KeyError: 'platform'
```

Use the `choices` parameter:

```
parser.add_argument('-f', '--fact', choices=['vendor', 'mgmt_ip', 'model', 'hostname', 'os'], help='enter a valid fact from the de
```

argparse - Using choices

```
ntc@jump-host:~$ python get_facts.py --f platform
usage: get_facts.py [-h] [-f {vendor,mgmt_ip,model,hostname,os}]
get_facts.py: error: argument -f/--fact: invalid choice: 'platform' (choose from 'vendor', 'mgmt_ip', 'model', 'hostname', 'os')
```

argparse - Using choices

```
ntc@jump-host:~$ python get_facts.py --f platform
usage: get_facts.py [-h] [-f {vendor,mgmt_ip,model,hostname,os}]
get_facts.py: error: argument -f/--fact: invalid choice: 'platform' (choose from 'vendor', 'mgmt_ip', 'model', 'hostname', 'os')
```

```
ntc@jump-host:~$ python get_facts.py -h
usage: get_facts.py [-h] [-f {vendor,mgmt_ip,model,hostname,os}]

Python Argparse Demo

optional arguments:
  -h, --help            show this help message and exit
  -f {vendor,mgmt_ip,model,hostname,os}, --fact {vendor,mgmt_ip,model,hostname,os}
                        enter a valid fact from the device facts dictionary
```

argparse - Multiple arguments

Objective:

- Pass in a fact you want to see the value for, but also include a description
- Code was modified to also print the description

```
ntc@jump-host:~$ python get_facts.py -h
usage: get_facts.py [-h] [-f {vendor,mgmt_ip,model,hostname,os}] [-d DESCR]

Python Argparse Demo

optional arguments:
  -h, --help            show this help message and exit
  -f {vendor,mgmt_ip,model,hostname,os}, --fact {vendor,mgmt_ip,model,hostname,os}
                        enter a valid fact from the device facts dictionary
  -d DESCR, --descr DESCR
                        enter a description for this job
```

```
ntc@jump-host:~$ python get_facts.py -f hostname -d "Test Job"
hostname: NYC301
Test Job
```

argparse - Adding descr argument

```python
import argparse

if __name__ == "__main__":

    facts = {'vendor': 'cisco', 'mgmt_ip': '10.1.1.1', 'model': 'nexus', 'hostname': 'NYC301', 'os': '6.1.2'}

    parser = argparse.ArgumentParser(description='Python Argparse Demo')
    parser.add_argument('-f', '--fact', help='enter a valid fact from the device facts dictionary')
    parser.add_argument('-d', '--descr', help='enter a description for this job')

    args = parser.parse_args()

    print(args.fact + ': ' + facts[args.fact])
    print(args.descr)
```

Summary

- For quick testing `sys.argv` is a great option
- For a more robust script, you want others to use and to have a more defined help menu, `argparse` is the way to go
    - Supports more features, feel free to continue to digging!

Lab Time

- Lab 20 - Passing in User Input
  - Prompt user input using `input` and process the input
  - Continue to build on the neighbors script from previous labs and only print certain neighbor and device information based on the arguments being passed in
  - Write a basic script using `sys.argv` that prints arguments

Regular Expressions

Python Network Libraries

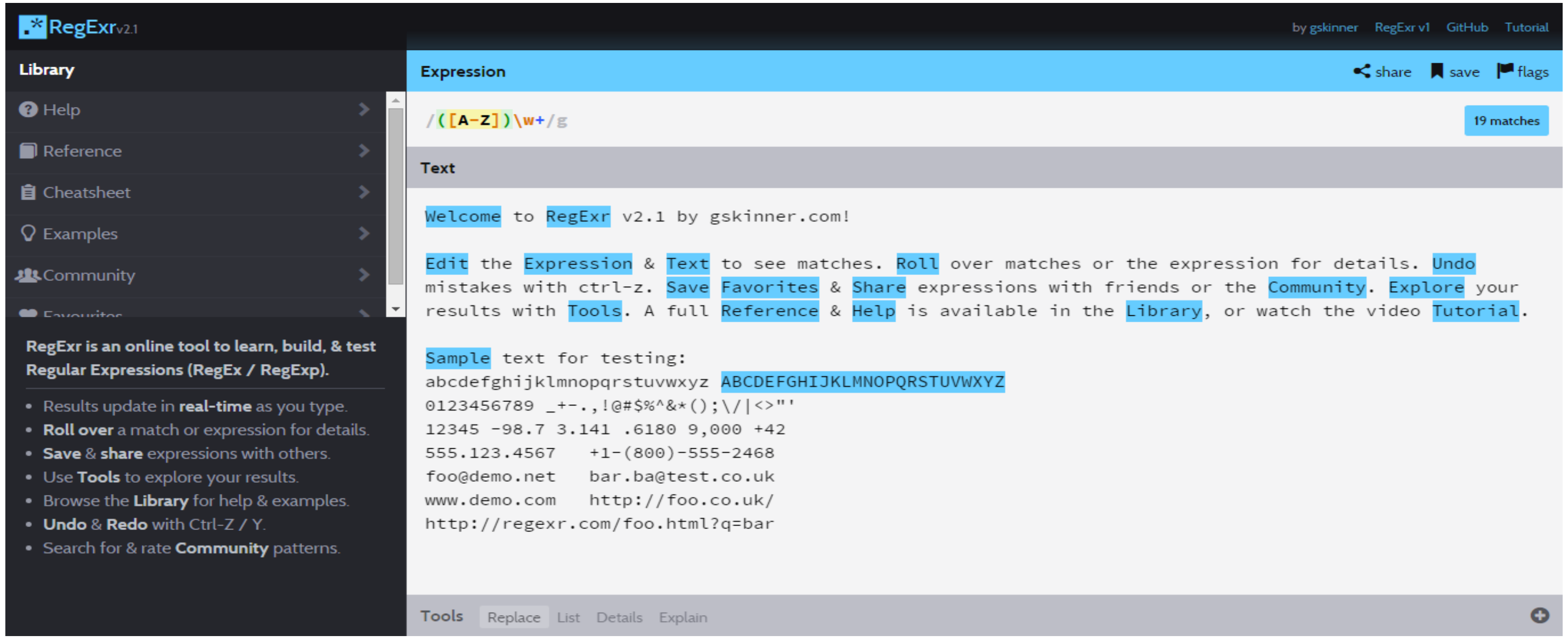
RegEx Overview

- A Regular Expression (RegEx) is a special sequence of characters used to search patterns inside text.
- They are a powerful tool for:
  - Checking if a specific pattern is present inside a text.
  - Parsing unstructured output from a network device.

Web Utilities for Testing

Online tools used for testing and learning regular expressions

- Regexr.com (picture below)
- Regex101.com



```
/([A-Z])\w+/g
```

RegExr is an online tool to learn, build, & test Regular Expressions (RegEx / RegExp).

- Results update in **real-time** as you type.
- **Roll over** a match or expression for details.
- **Save** & **share** expressions with others.
- Use **Tools** to explore your results.
- Browse the **Library** for help & examples.
- **Undo** & **Redo** with Ctrl-Z / Y.
- Search for & rate **Community** patterns.

```
Welcome to RegExr v2.1 by gskinner.com!

Edit the Expression & Text to see matches. Roll over matches or the expression for details. Undo
mistakes with ctrl-z. Save Favorites & Share expressions with friends or the Community. Explore your
results with Tools. A full Reference & Help is available in the Library, or watch the video Tutorial.

Sample text for testing:
abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789 _+-.,!@#$%^&*();\/|<>"'
12345 -98.7 3.141 .6180 9,000 +42
555.123.4567   +1-(800)-555-2468
foo@demo.net   bar.ba@test.co.uk
www.demo.com   http://foo.co.uk/
http://regexr.com/foo.html?q=bar
```

Regex patterns

- **\d**: Matches any decimal digit
- **\D**: Matches any non-digit character
- **\w**: Matches any alphanumeric character
- **\W**: Matches any non-alphanumeric character
- **\s**: Matches any whitespace character
- **\S**: Matches any non-whitespace character
- **.**: Matches anything except a newline character
- **+**: Specifies that the previous character can be matched one or more times
- ***** : Specifies that the previous character can be matched zero or more times
- **?** : Matches either once or zero times. Indicates something as being optional
  - Example: **ntc-?training** matches either **ntctraining** or **ntc-training**

Demo

- Review Regular Expressions using Regexr.com

TextFSM

Python Network Libraries

TextFSM Overview

- Python module for parsing semi-formatted text.
- Originally developed to allow programmatic access to information given by the output of CLI driven devices, such as network routers and switches
  - It can however be used for any such textual output.

Using TextFSM

- The engine takes two inputs
  - Template file
  - Text input (such as command responses from the CLI of device)
- Returns a list of records that contains the data parsed from the text.
- Note: A template file is needed for each uniquely structured text input.

TextFSM

Network Examples

Example 1: Text Input

- show vlan (Arista EOS)
- Filename: `arista_eos_show_vlan.raw`

```
VLAN  Name                             Status    Ports
----- -------------------------------- --------- -------------------------------
1     default                          active    Et1
10    Test1                            active    Et1, Et2
20    Test2                            suspended
30    VLAN0030                         suspended
```

Example 1: Template File

- show vlan (Arista EOS)
- Order is important
- Filename: `arista_eos_show_vlan.template`

```
Value VLAN_ID (\d+)
Value NAME (\w+)
Value STATUS (active|suspended)

Start
  ^${VLAN_ID}\s+${NAME}\s+${STATUS} -> Record
```

Example 1: Executing textfsm

```
VLAN  Name                             Status    Ports
----- -------------------------------- --------- -------------------------------
1     default                          active    Et1
```

```
ntc@jump-host$ python textfsm.py arista_eos_show_vlan.template arista_eos_show_vlan.raw
FSM Template:
Value VLAN_ID (\d+)
Value NAME (\w+)
Value STATUS (active|suspended)

Start
  ^${VLAN_ID}\s+${NAME}\s+${STATUS} -> Record


FSM Table:
['VLAN_ID', 'NAME', 'STATUS']
['1', 'default', 'active']
['10', 'Test1', 'active']
['20', 'Test2', 'suspended']
['30', 'VLAN0030', 'suspended']
```

Example 2: Text Input

- show version (Cisco IOS)
- Filename: `cisco_ios_show_version.raw`

```
Cisco IOS XE Software, Version 16.03.01
Cisco IOS Software [Denali], CSR1000V Software (X86_64LINUX_IOSD-UNIVERSALK9-M), Version 16.3.1, RELEASE SOFTWARE (fc3)
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2016 by Cisco Systems, Inc.
Compiled Tue 02-Aug-16 18:36 by mcpre

ROM: IOS-XE ROMMON

csr1 uptime is 2 minutes
Uptime for this control processor is 5 minutes
System returned to ROM by reload
System image file is "bootflash:packages.conf"
Last reload reason: reload

cisco CSR1000V (VXE) processor (revision VXE) with 2047392K/3075K bytes of memory.
Processor board ID 9KXI0D7TVFI
4 Gigabit Ethernet interfaces
32768K bytes of non-volatile configuration memory.
3984776K bytes of physical memory.
7774207K bytes of virtual hard disk at bootflash:.
0K bytes of  at webui:.

Configuration register is 0x2102
```

Example 3: Template File

- show version (Cisco IOS)
- Filename: `cisco_ios_show_version.template`

```
Value VERSION (.+?)
Value HOSTNAME (\S+)
Value UPTIME (.+)
Value CONFIG_REGISTER (\S+)

Start
  ^.*Software\s.+\),\sVersion\s${VERSION},*\s+RELEASE.*
  ^\s*${HOSTNAME}\s+uptime\s+is\s+${UPTIME}
  ^[Cc]onfiguration\s+register\s+is\s+${CONFIG_REGISTER} -> Record
```

Example 2: Executing textfsm

```
ntc@jump-host$ python textfsm.py cisco_ios_show_version.template cisco_ios_show_version.raw
FSM Template:
Value VERSION (.+?)
Value HOSTNAME (\S+)
Value UPTIME (.+)
Value CONFIG_REGISTER (\S+)

Start
  ^.*Software\s.+\),\sVersion\s${VERSION},*\s+RELEASE.*
  ^\s*${HOSTNAME}\s+uptime\s+is\s+${UPTIME}
  ^[Cc]onfiguration\s+register\s+is\s+${CONFIG_REGISTER}


FSM Table:
['VERSION', 'HOSTNAME', 'UPTIME', 'CONFIG_REGISTER']
['16.3.1', 'csr1', '2 minutes', '0x2102']
```

Using TextFSM in Python

From Example 1:

```
>>> import textfsm
>>>
>>> table = textfsm.TextFSM(open('arista_eos_show_vlan.template'))
>>>
>>> data = table.ParseText(open('arista_eos_show_vlan.raw').read())
>>>
>>>
>>> data
[['1', 'default', 'active'], ['10', 'Test1', 'active'], ['20', 'Test2', 'suspended'], ['30', 'VLAN0030', 'suspended']]
>>>
>>> table.header
['VLAN_ID', 'NAME', 'STATUS']
>>>
```

Using TextFSM in Python (cont'd)

From Example 2:

```
>>> import textfsm
>>>
>>> table = textfsm.TextFSM(open('cisco_ios_show_version.template'))
>>>
>>> data = table.ParseText(open('cisco_ios_show_version.raw').read())
>>> # data.table.ParseText(rawtext)
>>>
>>> data
[['16.3.1', 'csr1', '2 minutes', '0x2102']]
>>>
>>> table.header
['VERSION', 'HOSTNAME', 'UPTIME', 'CONFIG_REGISTER']
>>>
```

Summary

- Legacy devices are here to stay (for awhile)
- Even API-enabled device may return raw text
- Using TextFSM does not necessarily mean SSH/telnet as a transport mechanism
- Great way to bridge the gap between legacy and modern devices that return structured data

Managing TextFSM Templates

Python Network Libraries

clitable (TextFSM object)

- Builds upon textfsm object
- Simplifies use of pre-created templates
- Maps CLI commands to the proper template
    - Uses an index file and a templates directory
- Provides a better abstraction for consumers of templates

Templates Directory

- List of templates that `clitable` can use

```
cisco@ntc:~/projects/legacy/ntc_templates$ ls
arista_eos_show_clock.template                 cisco_ios_show_snmp_community.template
arista_eos_show_interfaces_status.template     cisco_ios_show_spanning-tree.template
arista_eos_show_ip_access-lists.template       cisco_ios_show_standby_brief.template
arista_eos_show_ip_arp.template                cisco_ios_show_vtp_status.template
arista_eos_show_ip_interface_brief.template    cisco_nxos_show_access-lists.template
arista_eos_show_lldp_neighbors.template        cisco_nxos_show_cdp_neighbors.template
arista_eos_show_mlag.template                  cisco_nxos_show_clock.template
arista_eos_show_snmp_community.template        cisco_nxos_show_feature.template
arista_eos_show_vlan.template                  cisco_nxos_show_flogi_database.template
cisco_ios_show_access-list.template            cisco_nxos_show_interface_brief.template
cisco_ios_show_cdp_neighbors.template          cisco_nxos_show_interface_status.template
cisco_ios_show_clock.template                  cisco_nxos_show_inventory.template
cisco_ios_show_interfaces_status.template      cisco_nxos_show_ip_arp_detail.template
cisco_ios_show_interfaces.template             cisco_nxos_show_ip_ospf_neighbor_vrf.template
cisco_ios_show_interface_transceiver.template  cisco_nxos_show_ip_route.template
cisco_ios_show_inventory.template              cisco_nxos_show_lldp_neighbors.template
cisco_ios_show_ip_arp.template                 cisco_nxos_show_mac_address-table.template
cisco_ios_show_ip_bgp_summary.template         cisco_nxos_show_port-channel_summary.template
cisco_ios_show_ip_bgp.template                 cisco_nxos_show_version.template
cisco_ios_show_ip_int_brief.template           cisco_nxos_show_vlan.template
cisco_ios_show_ip_ospf_neighbor.template       cisco_nxos_show_vpc.template
cisco_ios_show_ip_route.template               cisco_wlc_ssh_show_sysinfo.template
cisco_ios_show_lldp_neighbors.template         hp_comware_display_vlan_brief.template
cisco_ios_show_mac-address-table.template      index
```

Index File

- Maps Template to **Platform** specific **Command**
    - Can get more granular by specifying **Hostname** too

```
ntc@jump-host:~/projects/legacy$ more ntc_templates/index

# First line is the header fields for columns and is mandatory.
# Regular expressions are supported in all fields except the first.
# Last field supports variable length command completion.
# abc[[xyz]] is expanded to abc(x(y(z)?)?)?, regexp inside [[]] is not supported
#
Template, Hostname, Platform, Command
cisco_nxos_show_vlan.template, .*, cisco_nxos, sh[[ow]] vl[[an]]
cisco_ios_show_ip_int_brief.template, .*, cisco_ios, sh[[ow]] ip int[[erface]] br[[ief]]
cisco_nxos_show_ip_route.template, .*, cisco_nxos, sh[[ow]] ip route
hp_comware_display_vlan_brief.template, .*, hp_comware, di[[splay]] v[[lan]] b[[rief]]
cisco_nxos_show_version.template, .*, cisco_nxos, sh[[ow]] ver[[sion]]
cisco_wlc_ssh_show_sysinfo.template, .*, cisco_wlc_ssh, sh[[ow]] sysi[[nfo]]
cisco_ios_show_ip_ospf_neighbor.template, .*, cisco_ios, sh[[ow]] ip ospf nei[[ghbor]]
cisco_nxos_show_feature.template, .*, cisco_nxos, sh[[ow]] feat[[ure]]
arista_eos_show_vlan.template, .*, arista_eos, sh[[ow]] vl[[an]]
cisco_nxos_show_mac_address-table.template, .*, cisco_nxos, sh[[ow]] m[[ac]] addr[[ess-table]]
cisco_ios_show_snmp_community.template, .*, cisco_ios, sh[[ow]] sn[[mp]] com[[munity]]
cisco_ios_show_access-list.template, .*, cisco_ios, sh[[ow]] acc[[ess-list]]
arista_eos_show_clock.template, .*, arista_eos, sh[[ow]] cl[[ock]]
```

Using clitable in Python

```python
import clitable

index_file = 'index'
template_dir = '/etc/ntc/ansible/library/ntc-ansible/ntc-templates/templates'

cli_table = clitable.CliTable(index_file, template_dir)

command = 'show vlan'
platform = 'cisco_nxos'

# keys map directly back to column headers in the index file (see previous slide)
attrs = {'Command': command, 'Platform': platform}

# rawtxt is the show output as a string; could be from a file or from device in real-time
cli_table.ParseCmd(rawtxt, attrs)

print(cli_table)
```

Using clitable in Python

```python
import clitable

index_file = 'index'
template_dir = '/etc/ntc/ansible/library/ntc-ansible/ntc-templates/templates'

cli_table = clitable.CliTable(index_file, template_dir)

command = 'show vlan'
platform = 'cisco_nxos'

# keys map directly back to column headers in the index file (see previous slide)
attrs = {'Command': command, 'Platform': platform}

# rawtxt is the show output as a string; could be from a file or from device in real-time
cli_table.ParseCmd(rawtxt, attrs)

print(cli_table)
```

```
>>> print(cli_table)
VLAN_ID, NAME, STATUS
1, default, active
10, Test1, active
20, Test2, suspended
30, VLAN0030, suspended

>>> type(cli_table)
<class 'clitable.CliTable'>
```

Simplifying CliTable Objects

- `clitable_to_dict()`
- Creates list of dictionaries from a `CliTable` object
- For commands like `show version`, it still creates a list of one element

```
>>> from ntc_course import clitable_to_dict
>>>
>>> help(clitable_to_dict)
>>>

Help on function clitable_to_dict in module ntc_course:

clitable_to_dict(cli_table)
    Converts TextFSM cli_table object to list of dictionaries
    Borrowed for this course from the ntc-ansible project at
    github.com/networktocode/ntc-ansible/library/ntc_show_command
(END)
```

Covert CliTable to List of Dictionaries

```
>>> print(cli_table)
VLAN_ID, NAME, STATUS
1, default, active
10, Test1, active
20, Test2, suspended
30, VLAN0030, suspended
>>>
>>> type(cli_table)
<class 'clitable.CliTable'>
```

```
>>> from ntc_course import clitable_to_dict
>>>
>>> clitable_to_dict(cli_table)
[{'vlan_id': '1', 'name': 'default', 'status': 'active'},
{'vlan_id': '10', 'name': 'Test1', 'status': 'active'},
{'vlan_id': '20', 'name': 'Test2', 'status': 'suspended'},
{'vlan_id': '30', 'name': 'VLAN0030', 'status': 'suspended'}]
```

Summary

- Level of abstraction above textfsm.py
- clitable makes it you don't have to know which template you need to call
- Rather, you know the command and Command and Platform

Lab Time

- Lab 21 - Parsing Show Commands with TextFSM
  - Use TextFSM to parse `show ip interface brief` from a Cisco Nexus switch
  - Use `clitable` along with `netmiko` to generate structured data from unstructured device output
  - **Note: same workflow and process can be used for any other device.**