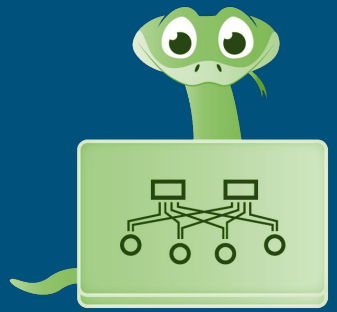


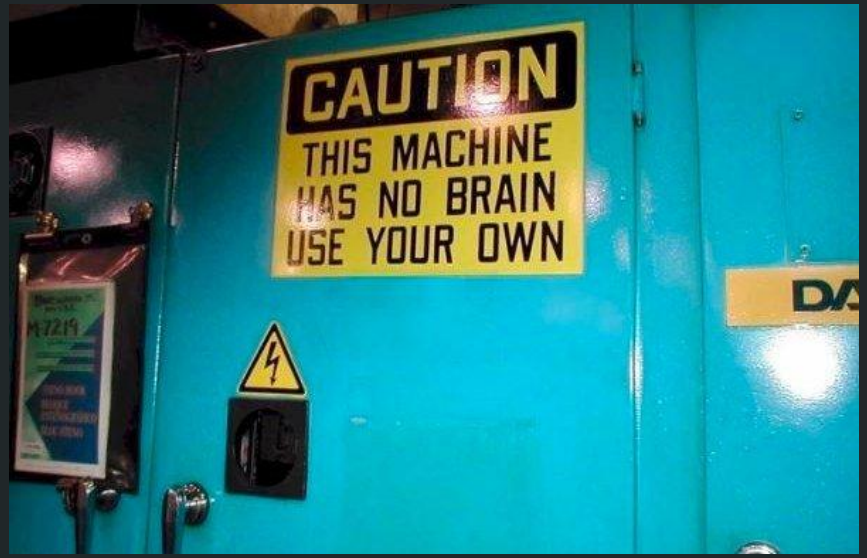
PYTHON

FOR NETWORK ENGINEERS

Onsite Training Session
July 2020

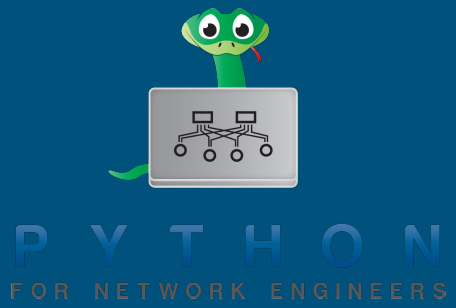
Day3 Schedule

- Pdb - The Python Debugger
- Juniper PyEZ Configurations
- Juniper PyEZ Direct RPC
- XML
- SSH and Concurrency
- Requests and using a Rest API (NetBox)



Flickr: Pierre-Olivier Carles

Review Exercise



1. Load the ~/.netmiko.yml and use this to connect with Netmiko to all of the devices in the lab environment.

Pdb - The Python Debugger

```
python -m pdb my_script.py

import pdb
pdb.set_trace()
```

<u>Pdb Commands</u>

```
help (h)
list (l)
list 1          # list starting at line1
list 1, 25      # list lines 1 - 25

next (n)        # Step one line at a time; don't descend
step (s)        # Step one line at a time descend into callables
break 16 (b 16) # Set a breakpoint at line 16
continue (c)    # Continue execution
```

Pdb - The Python Debugger

./day3/pdb/pdb_ex1.txt

Pdb Commands

```
down (d)        # Move down the stack
up (u)          # Move up the stack

p foo           # Print out variable foo
pp foo          # Pretty print out variable foo

!print("hello")  # Exclamation point can prefix generic Python code

quit (q)        # Abort the current Pdb session and program
```

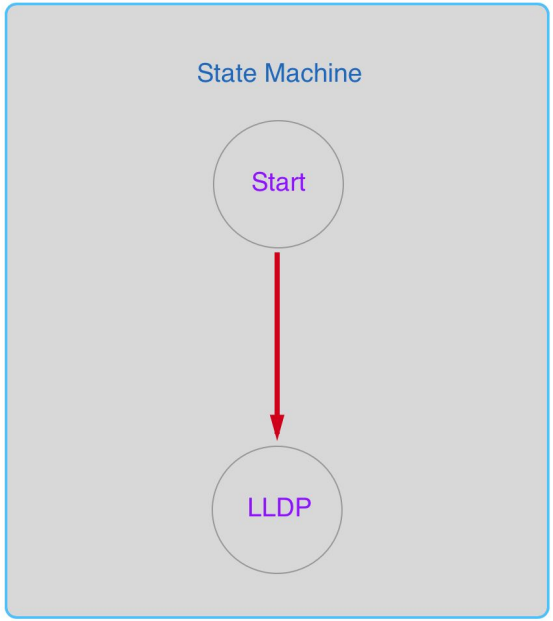
PYTHON
FOR NETWORK ENGINEERS

TextFSM - The Problem

```
$ cat show_ip_bgp.txt
BGP table version is 17889841, local router ID is 128.223.51.103
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure, S Stale, m multipath, b backup-path, f RT-Filter,
              x best-external, a additional-path, c RIB-compressed,
Origin codes: i - IGP, e - EGP, ? - incomplete
RPKI validation codes: V valid, I invalid, N Not found

     Network          Next Hop            Metric LocPrf Weight Path
 *   1.0.0.0/24       208.74.64.40                           0 19214 174 13335 i
 *                    162.251.163.2                          0 53767 13335 i
 *                    94.142.247.3             0             0 8283 13335 i
 *                    212.66.96.126                          0 20912 13335 i
```

TextFSM



TextFSM File

```
# Define your fields to extract
Value VAR_NAME (regex_pattern)
Value VAR_NAME (regex_pattern)
Value VAR_NAME (regex_pattern)


# Start of the FSM
Start
  ^Device.*ID -> LLDP

LLDP
  ^${VAR_NAME}.* -> Record

# Implicit EOF and Record
# EOF
```

TextFSM - A minimum set of regular expressions

Regular Expression Special Chars

```
\d      Digits 0-9
\s      Whitespace characters
\S      Non-whitespace
\w      Alphanumeric includes _

.       Any single character

*       Repeated 0 or more times
+       Repeated 1 or more times

^       Beginning of the line anchor
$       End of the line anchor


Greedy by-default.
```

TextFSM - Example

Reference Material in:
{{ github_repo }}/textfsm

- Variables > Start > State Transition
- Implicit EOF
- Installing TextFSM
- Installing ntc-templates
- Coupling TextFSM with Netmiko

PyEZ config operations

Exercises:
./day3/jnpr/ex3.txt

```
a_device = Device(host="srx2.lasthop.io", user="pyclass", password=getpass())
a_device.open()
a_device.timeout = 60

cfg = Config(a_device)
cfg.lock()

cfg.load("set system host-name test123", format="set", merge=True)
cfg.rollback(0)

cfg.load("set system host-name test123", format="set", merge=True)
print(cfg.diff())

cfg.commit()
```

PyEZ direct RPC

```
pyclass@vmx2> show version | display xml rpc
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/18.4R1/junos">
    <rpc>
        <get-software-information>
        </get-software-information>
    </rpc>
    <cli>
        <banner></banner>
    </cli>
</rpc-reply>
```

PyEZ direct RPC

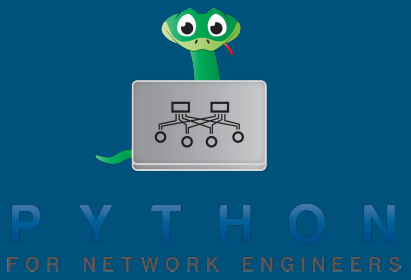
```
a_device = Device(**device)
a_device.open()

# show version | display xml rpc
# <get-software-information>
xml_out = a_device.rpc.get_software_information()
print(etree.tostring(xml_out, encoding="unicode", pretty_print=True))
```

PyEZ direct RPC (XML)

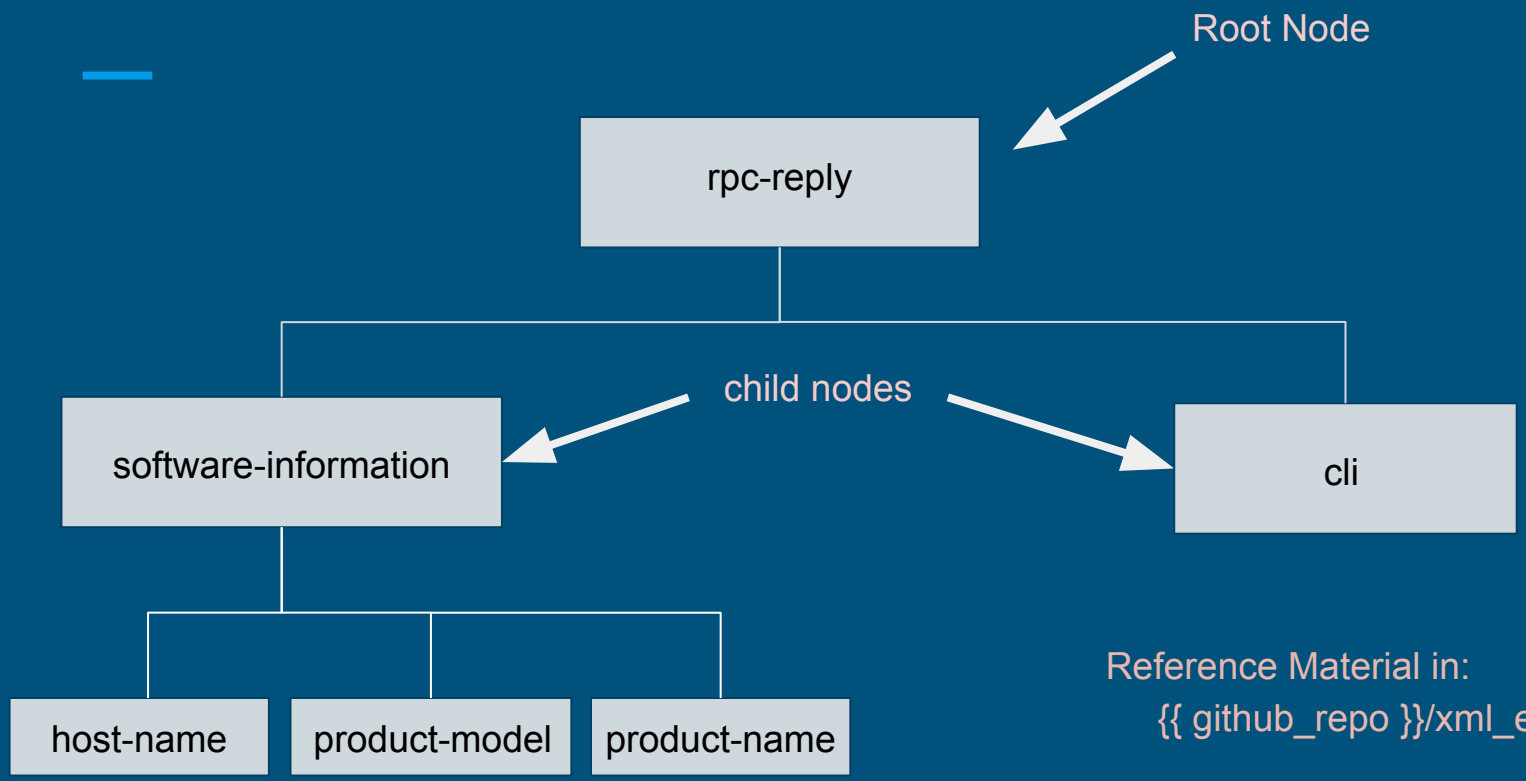

```
<software-information>
  <host-name>vmx1</host-name>
  <product-model>vmx</product-model>
  <product-name>vmx</product-name>
  <junos-version>18.4R1.8</junos-version>
  <package-information>
    <name>os-kernel</name>
    <comment>JUNOS OS Kernel 64-bit  [20181207.6c2f68b_2_builder_stable_11]</comment>
  </package-information>
  <package-information>
    <name>os-libs</name>
    <comment>JUNOS OS libs [20181207.6c2f68b_2_builder_stable_11]</comment>
  </package-information>
  <package-information>
    <name>os-runtime</name>
    <comment>JUNOS OS runtime [20181207.6c2f68b_2_builder_stable_11]</comment>
```

XML: the good, the bad, and the ugly




```
pyclass@vmx1> show version | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/18.4R1/junos">
    <software-information>
        <host-name>vmx1</host-name>
        <product-model>vmx</product-model>
        <product-name>vmx</product-name>
        <junos-version>18.4R1.8</junos-version>
        <package-information>
            <name>os-kernel</name>
            <comment>JUNOS OS Kernel 64-bit  [20181207.6c2f68b_2_builder_stable_11]</comment>
        </package-information>
        <package-information>
            <name>os-libs</name>
            <comment>JUNOS OS libs [20181207.6c2f68b_2_builder_stable_11]</comment>
        </package-information>
        <package-information>
            <name>os-runtime</name>
            <comment>JUNOS OS runtime [20181207.6c2f68b_2_builder_stable_11]</comment>
```

XML - Think of it as a tree of nodes



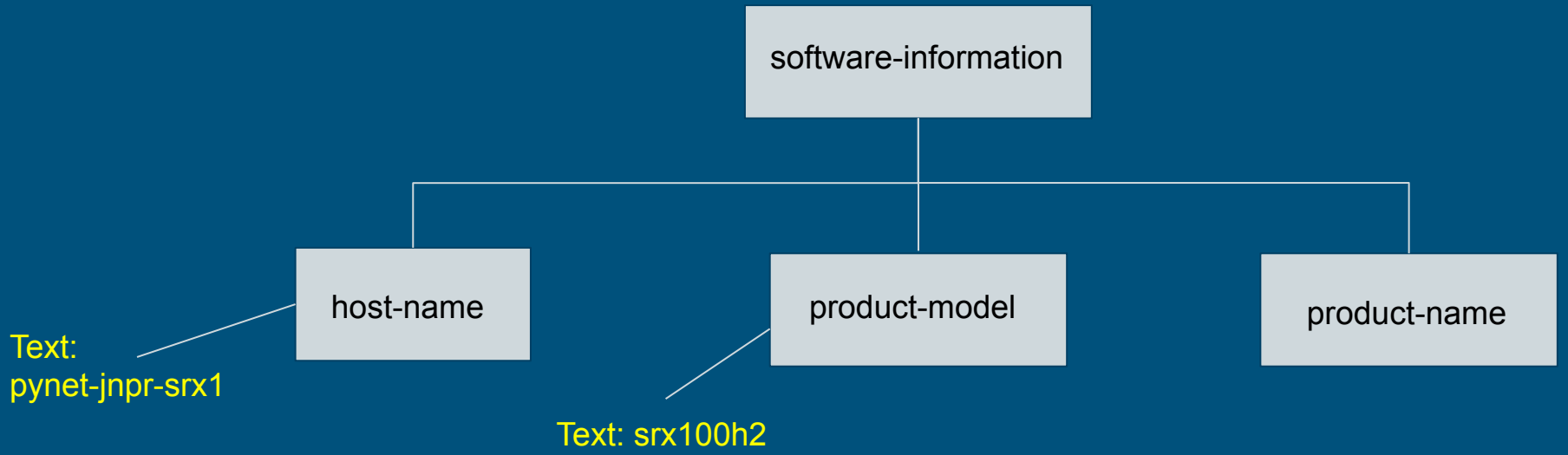
Reference Material in:
{{ github_repo }}/xml_example

XML Text “Nodes”

```
> show version | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/12.1X44/junos">
    <software-information>
        <host-name>pynet-jnpr-srx1</host-name>
        <product-model>srx100h2</product-model>
        <product-name>srx100h2</product-name>
        <jsr/>
        <package-information>
            <name>junos</name>
            <comment>JUNOS Software Release [12.1X44-D35.5]</comment>
        </package-information>
    </software-information>
    <cli>
        <banner></banner>
    </cli>
</rpc-reply>
```

XML Text “Nodes” (ElementTree/lxml Perspective)

Treat the Text as an attribute of the Element Node

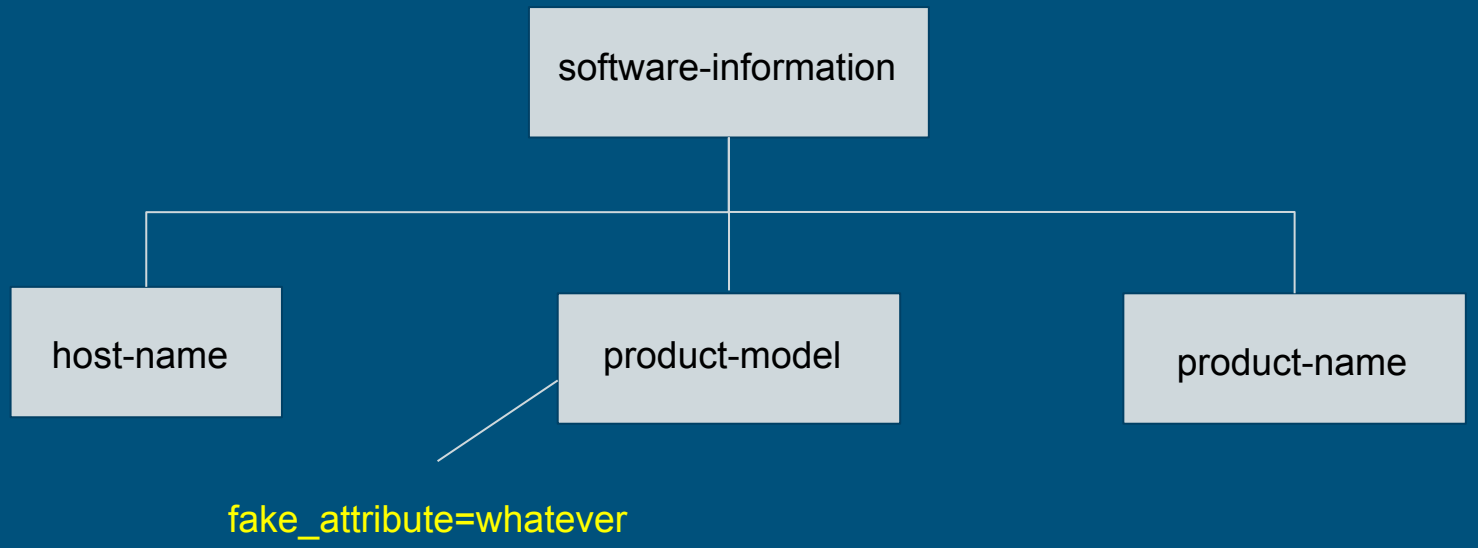


XML Attribute “Nodes”

```
> show version | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/12.1X44/junos">
    <software-information>
        <host-name>pynet-jnpr-srx1</host-name>
        <product-model fake_attribute=’whatever’>srx100h2</product-model>
        <product-name>srx100h2</product-name>
        <jsr/>
        <package-information>
            <name>junos</name>
            <comment>JUNOS Software Release [12.1X44-D35.5]</comment>
        </package-information>
    </software-information>
    <cli>
        <banner></banner>
    </cli>
</rpc-reply>
```

XML Attribute “Nodes” (ElementTree/lxml Perspective)

Treat the Attribute as an attribute of the Element Node



This is not what the DOM does?

In the DOM (document object model):

The following are nodes (and other things are also nodes):

- Element Nodes
- Text Nodes
- Attribute Nodes

Implications of this when using Python

Terminology: Element

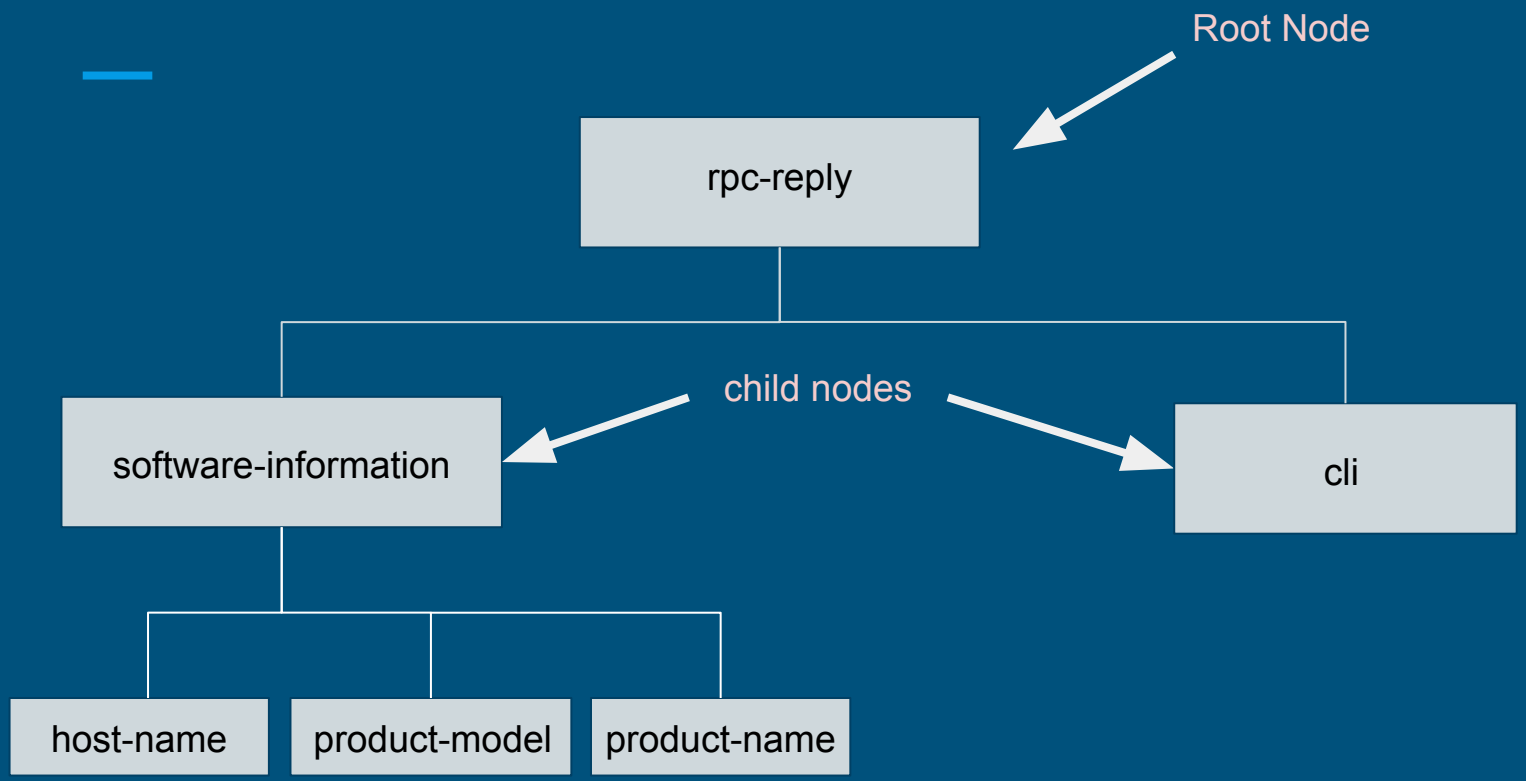
```
> show version | display xml
<rpc-reply xmlns:junos="http://xml.juniper.net/junos/12.1X44/junos">
    <software-information>
        <host-name>pynet-jnpr-srx1</host-name>
        <product-model>srx100h2</product-model>
        <product-name>srx100h2</product-name>
        <jsr/>
        <package-information>
            <name>junos</name>
            <comment>JUNOS Software Release [12.1X44-D35.5]</comment>
        </package-information>
    </software-information>
    <cli>
        <banner></banner>
    </cli>
</rpc-reply>
```

Terminology:

Child Nodes
Parent Nodes
Sibling Nodes
Ancestor Nodes
Descendant Nodes

Namespaces - a way to uniquely identify the names of nodes.

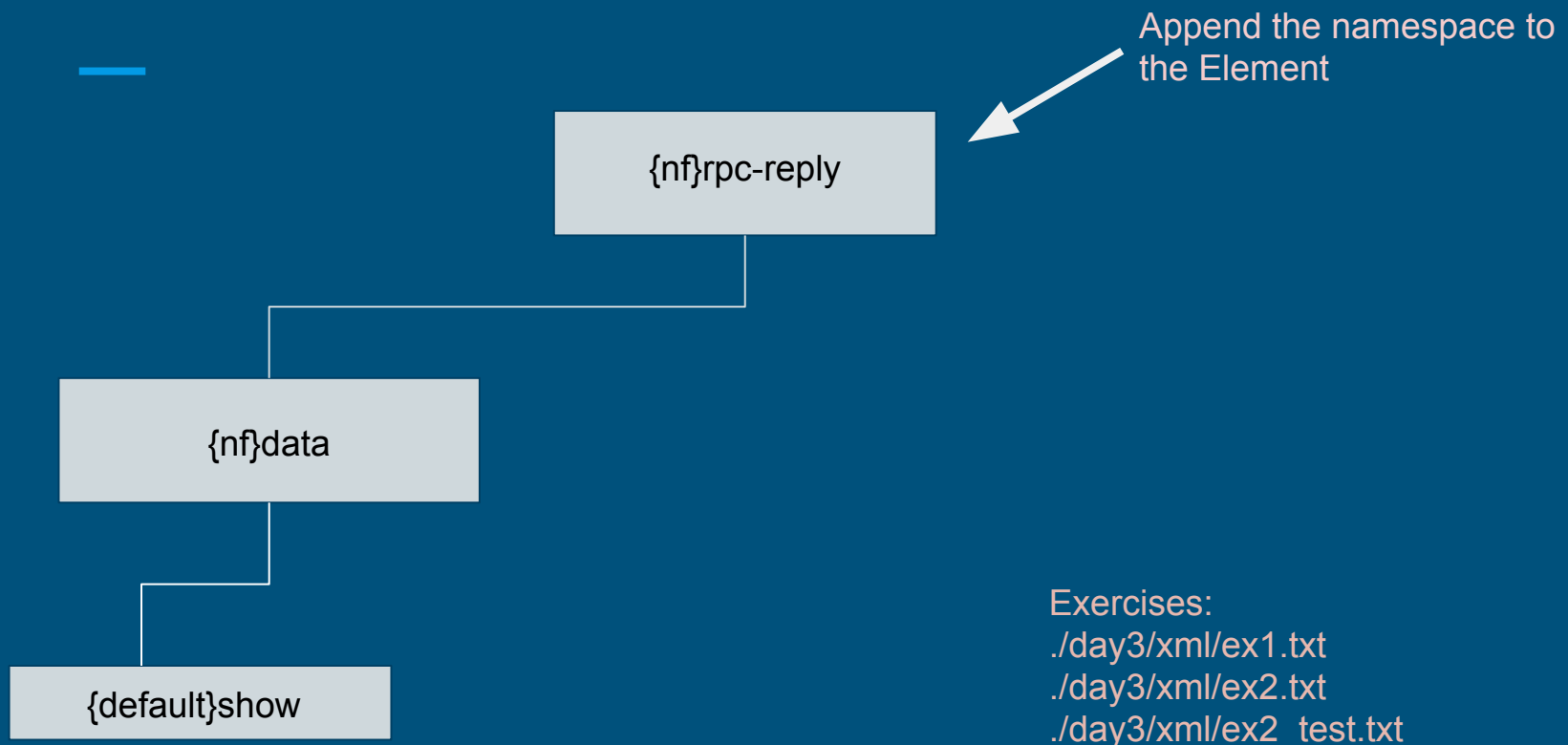
XML


Root Node
rpc-reply
child nodes
software-information
cli
host-name
product-model
product-name

Namespaces

```
nxos1# show version | xml
<?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:sysmgrcli">
 <nf:data>
  <show>
   <version>
       ...
   </version>
  </show>
 </nf:data>
</nf:rpc-reply>
```

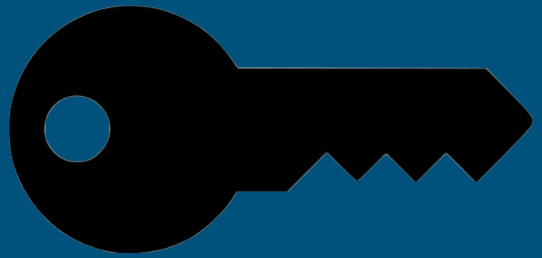
XML with Namespaces



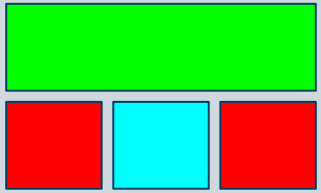
Exercises:
./day3/xml/ex1.txt
./day3/xml/ex2.txt
./day3/xml/ex2_test.txt

Concurrency/Parallelism

- Concurrency? Parallelism?
- Python and the GIL
- Concurrent Futures



Concurrency



Parallelism

Concurrent Futures

- Python 3.2 + backported to Python 2
- Wrapper around Threading/Processes
- Provides consistent interface using either Threads or Processes -- meaning very easy to switch concurrency method
- Threads: for I/O bound things (waiting for stuff in the network)
- Processes: for CPU bound things (crunch lots and lots of numbers)

Concurrent Futures - ThreadPool

Reference Material in:

{{ github_repo }}/concurrency_example

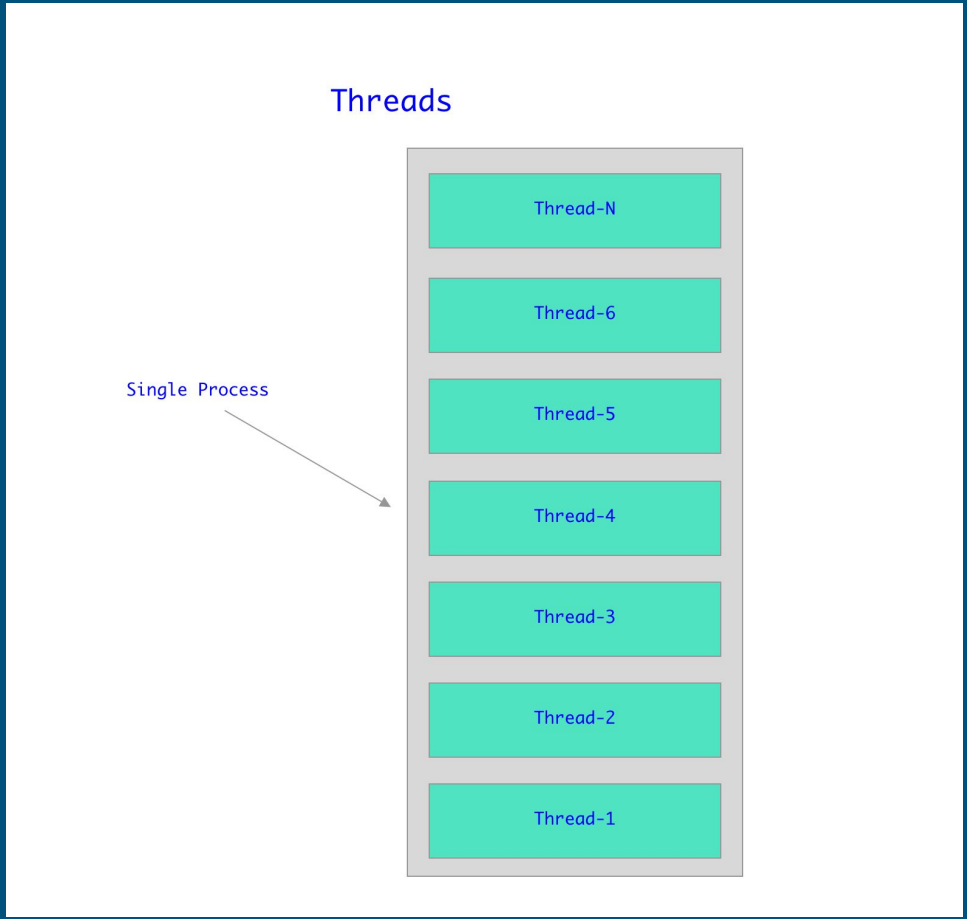
Exercises:
./day3/concurrency/ex1.txt
./day3/concurrency/ex1_testing.txt

```
def main():
    # Create your thread pool
    pool = ThreadPoolExecutor(max_workers=WORKERS)
    futures = []

    # Submit the work to the thread pool
    for _ in range(TASKS):
        futures.append(pool.submit(math_calculation))

    # 'wait' will block until all of the tasks are complete
    wait(futures)
    for task_result in futures:
        print(task_result.result())
```

Concurrent Futures - ThreadPool



Concurrent Futures - ProcessPool

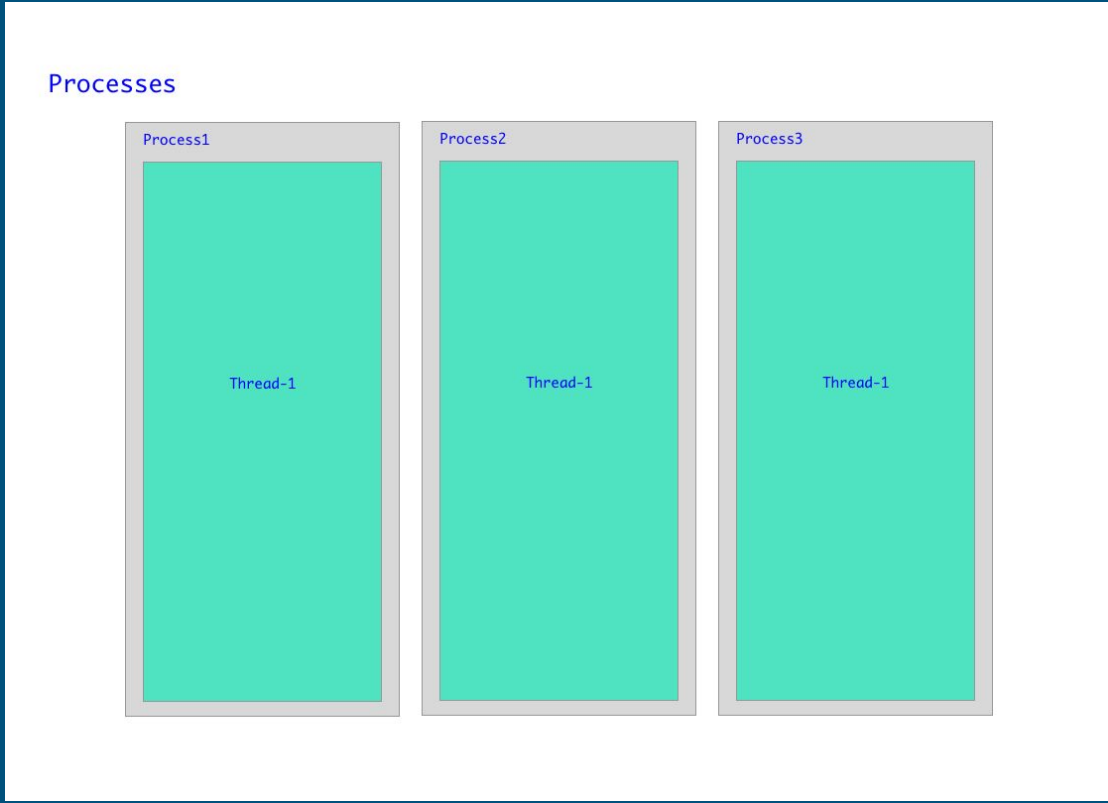
Exercises:
./day3/concurrency/ex2.txt
./day3/concurrency/ex2_testing.txt

```
def main():
    # Create process pool
    pool = ProcessPoolExecutor(max_workers=PROC_POOL)
    futures = []

    # Submit work to process pool
    for _ in range(TASKS):
        futures.append(pool.submit(math_calculation))

    # Block waiting for tasks to complete
    wait(futures)
    for task_result in futures:
        print(task_result.result())
```

Concurrent Futures - ProcessPool



Concurrent Futures - As Completed

```
def main():
    # Create process pool
    pool = ProcessPoolExecutor(max_workers=PROC_POOL)
    futures = []

    # Submit work to process pool
    for _ in range(TASKS):
        futures.append(pool.submit(math_calculation))

    # Show results as the work completes
    for task_result in as_completed(futures):
        print(task_result.result())
```

REST API

Not Secure | https://netbox.lasthop.io/api/

Apps | SDN Live Seminar... | View pynet Discus... | Python Net Engr | Web Design | Weekly Zag Briefi... | Read Later | Network Engineeri... | https://github.co... | Email software | Tools | Engines | Blog

NetBox

API Root

API Root

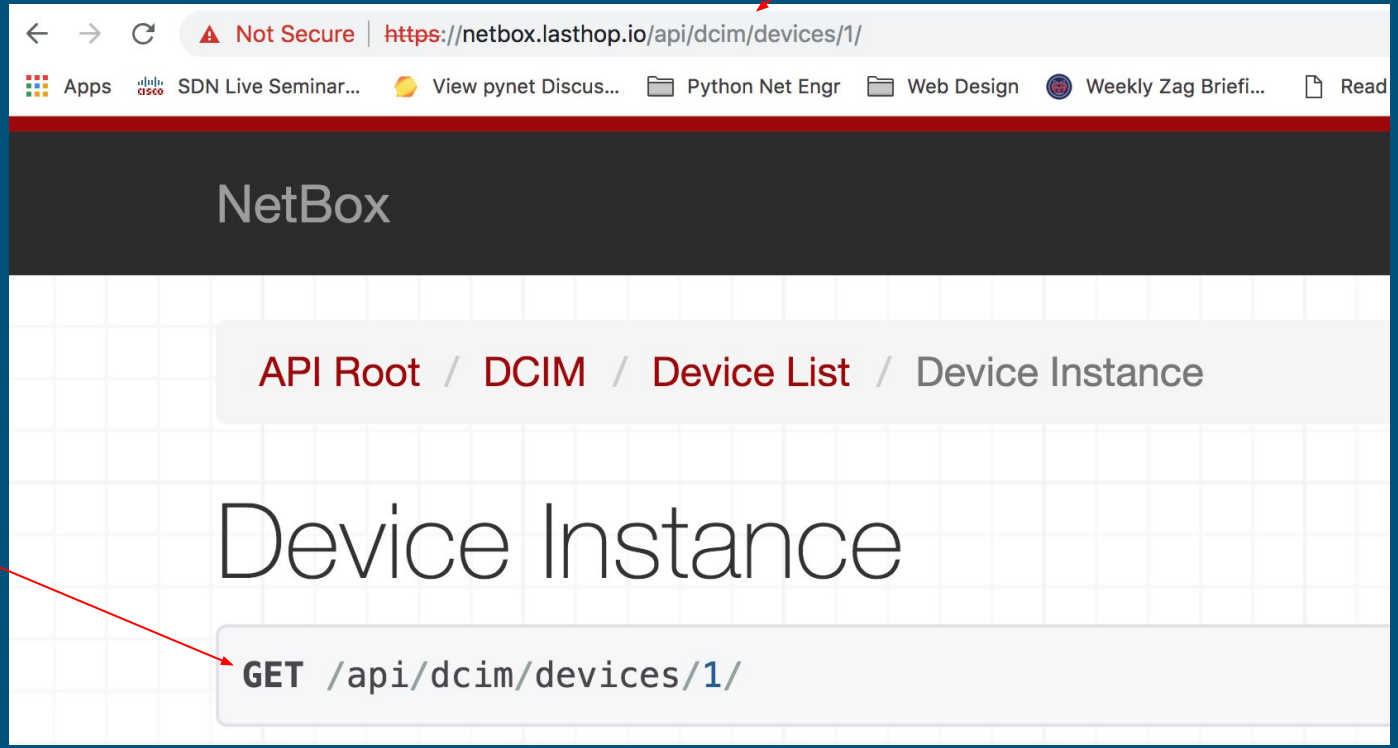
```
GET /api/
```

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "circuits": "http://netbox.lasthop.io/api/circuits/",
    "dcim": "http://netbox.lasthop.io/api/dcim/",
    "extras": "http://netbox.lasthop.io/api/extras/",
    "ipam": "http://netbox.lasthop.io/api/ipam/",
    "secrets": "http://netbox.lasthop.io/api/secrets/",
    "tenancy": "http://netbox.lasthop.io/api/tenancy/",
    "virtualization": "http://netbox.lasthop.io/api/virtualization/"
}
```

REST API - Characteristics

URL - the object I am accessing.



HTTP Method

REST API - Other HTTP Methods

Available HTTP Methods

API Root / DCIM / Device List / Device Instance

Device Instance

```
GET /api/dcim/devices/1/
```

```
HTTP 200 OK
Allow: GET, PUT, PATCH, DELETE, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept
```

REST API - CRUD

- Create - HTTP Post
- Read - HTTP Get
- Replace - HTTP Put
- Update - HTTP Patch
- Delete - HTTP Delete

Remember: Not all APIs are the same!

HTTP Methods Ref: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

REST API - Accessing API via Browser + CLI

```
(py3_venv) [kbyers@ip-172-30-0-118 ~]$
(py3_venv) [kbyers@ip-172-30-0-118 ~]$ curl -s https://netbox.lasthop.io/api/ --insecure | jq "."
{
  "circuits": "http://netbox.lasthop.io/api/circuits/",
  "dcim": "http://netbox.lasthop.io/api/dcim/",
  "extras": "http://netbox.lasthop.io/api/extras/",
  "ipam": "http://netbox.lasthop.io/api/ipam/",
  "secrets": "http://netbox.lasthop.io/api/secrets/",
  "tenancy": "http://netbox.lasthop.io/api/tenancy/",
  "virtualization": "http://netbox.lasthop.io/api/virtualization/"
}
(py3_venv) [kbyers@ip-172-30-0-118 ~]$
```

REST API - Basic Requests Get

Reference Material in:
{{ github_repo }}/rest_api

```python
import requests
from pprint import pprint

from urllib3.exceptions import InsecureRequestWarning

requests.packages.urllib3.disable_warnings(category=InsecureRequestWarning)


if __name__ == "__main__":

    url = "https://netbox.lasthop.io/api/dcim/"
    # url = "https://api.github.com/"
    http_headers = {"accept": "application/json; version=2.4;"}
    response = requests.get(url, headers=http_headers, verify=False)
    response = response.json()

    print()
    pprint(response)
    print()
```

Authentication

- Simple Auth
- Token Based
- OAuth

```python
import requests
from pprint import pprint

from urllib3.exceptions import InsecureRequestWarning

requests.packages.urllib3.disable_warnings(category=InsecureRequestWarning)


if __name__ == "__main__":

    token = "1234123412341234123412341341341134123433"
    url = "https://netbox.lasthop.io/api/dcim/devices/1"
    http_headers = {"accept": "application/json; version=2.4;"}
    if token:
        http_headers["authorization"] = "Token {}".format(token)

    response = requests.get(url, headers=http_headers, verify=False)
    response = response.json()

    print()
    pprint(response)
    print()
```

Tokens, Tokens Everywhere! & REST API - Basic Requests POST

- Tokens can be included in multiple locations:
  - Headers
  - Encoded in URL
  - In a payload

- POST - expects a “payload”

```python
def main():
    # Get list of channels; authenticate with token in the header
    headers = {"Authorization": f"Bearer {SLACK_TOKEN}"}
    resp = requests.get(f"{SLACK_BASE_URL}/channels.list", headers=headers)
    pprint(resp.json())

    print()

    # Get list of channels; authenticate with token encoded in url
    resp = requests.get(f"{SLACK_BASE_URL}/channels.list?token={SLACK_TOKEN}")
    pprint(resp.json())

    print()

    # Get list of channels; authenticate with token encoded in url
    data = {"token": SLACK_TOKEN}
    resp = requests.post(f"{SLACK_BASE_URL}/channels.list", data=data)
    pprint(resp.json())

    print()
```

REST API - Adding a device using HTTP POST

netbox | Organization | Racks | Devices | IPAM | Virtualization | Circuits | Secrets

+ Add | Import | Expor

Devices

Name	Status	Tenant	Site	Rack	Role	Type	IP Address
arista1	Active	—	Fremont Data Center	RK1	Distribution Switch	Arista vEOS	184.105.247.72
arista2	Active	—	Fremont Data Center	RK1	Distribution Switch	Arista vEOS	
arista3	Active	—	Fremont Data Center	RK1	Distribution Switch	Arista vEOS	
arista4	Active	—	Fremont Data Center	RK1	Distribution Switch	Arista vEOS	
arista5	Active	—	Fremont Data Center	RK1	Distribution Switch	Arista vEOS	
arista6	Active	—	Fremont Data Center	RK2	Distribution Switch	Arista vEOS	
arista7	Active	—	Fremont Data Center	RK1	Distribution Switch	Arista vEOS	

REST API - Adding a device using HTTP POST

```
http_headers = {
    "Content-Type": "application/json; version=2.4;",
    "authorization": "Token {}".format(token),
}
post_data = {
    "name": "arista8",
    "device_role": 3,  # Distribution Switch
    "device_type": 2,  # vEOS
    "display_name": "arista8",
    "platform": 4,  # Arista EOS
    "rack": 1,  # RK1
    "site": 1,  # Fremont Data Center
    "status": 1,  # Active
}

response = requests.post(
    url, headers=http_headers, data=json.dumps(post_data), verify=False
)
response = response.json()
```

REST API - Modify (put) and Delete

Exercises:
./day3/rest/ex1.txt
./day3/rest/ex2.txt
./day3/rest/ex3.txt

```
response = requests.put(
    url, headers=http_headers, data=json.dumps(arista6), verify=False
)
```

```
response = requests.delete(url, headers=http_headers, verify=False)
```