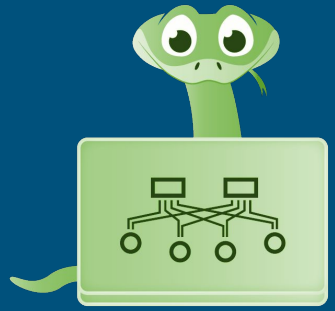


# PYTHON

## FOR NETWORK ENGINEERS

Onsite Training Session
November 2019

# $ whoami

Kirk Byers
Network Engineer:
CCIE #6243 (emeritus)

Programmer:
Netmiko
NAPALM
Nornir

Teach Python, Ansible, Nornir in a Network Automation context

# $ whoami

Carl Montanari
Network Engineer:
Dual-CCIE (#37652 R/S and SP)

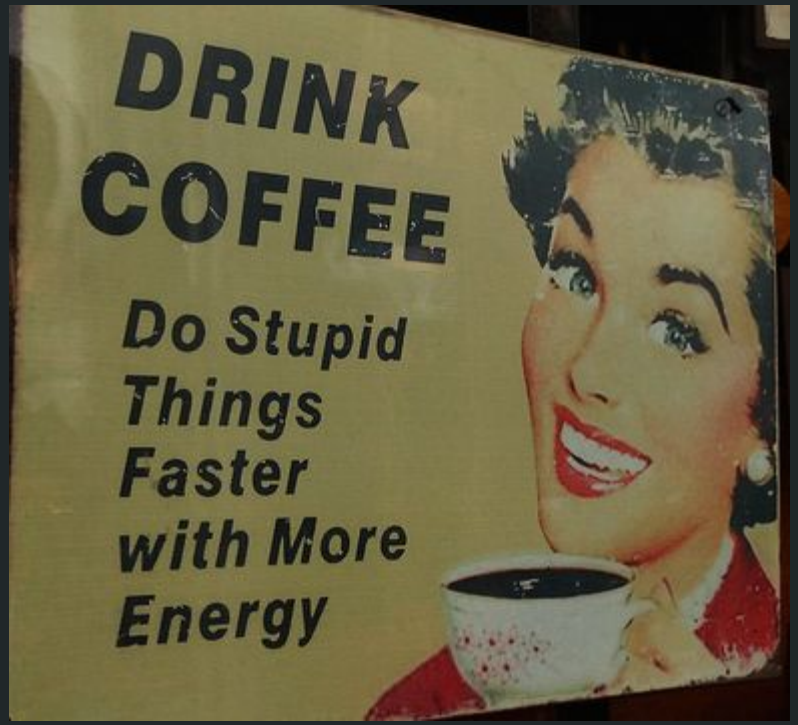
R/S and Data Center focus, then down the automation rabbit hole!

Help folks to learn and implement network automation

# General:

November 19, 9:00AM - 5:00PM
November 20, 9:00AM - 5:00PM
November 21, 9:00AM - 5:00PM
November 22, 9:00AM - 5:00PM

Focused/Minimize Distractions
Exercises and Examples
Examples in the Python Shell
Try not to fall behind on day 1 & 2!



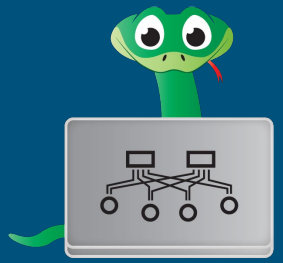

Flickr: Ben Sutherland

# Course Overview

- Day 1 - The Foundational Bits
  - Git
  - Why Python
  - Python Fundamentals
- Day 2 - Connecting to Devices
  - Regex
  - Netmiko, Netmiko, Netmiko!
  - Python LEGOs (reuse functions, classes, objects)
- Day 3 - Data and APIs
  - Data serialization
  - Parsers: TextFSM, Genie
  - APIs: NX-API, REST-API
- Day 4 -
  - Templating (seriously, a super power)
  - NAPALM
  - Concurrency

# Day 1 Schedule

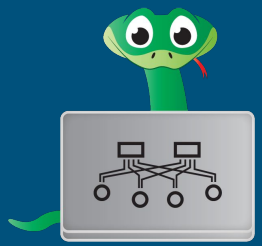
- vi in Five Minutes(ish)
- Working with Git
- Why Python?
- Python Fundamentals
  - Strings
  - Lists, Dictionaries
  - Files
  - Conditionals
  - Loops
  - Exceptions
  - And more!



PYTHON

FOR NETWORK ENGINEERS

# vi in Five Minutes(ish)



PYTHON
FOR NETWORK ENGINEERS

SSH into lab environment

vi test1.txt

Two modes: edit-mode and command-mode (ESC is your path to safety).

i - insert (switch to edit-mode)
a - append (switch to edit-mode)

Never, absolutely never, hit caps-lock it is the path to destruction and ruin.

Use h, j, k, l to navigate (in command-mode)

**Note:** type `vimtutor` at the lab host prompt to be dropped into an interactive tutorial for vi!

# vi in Five Minutes(ish)

Use h, j, k, l to navigate (in command-mode)

h - move left one space
j - move down one space
k - move up one space
l - move right one space

Arrow keys will also probably work.

x - delete a character
dw - delete a word
dd - delete a line

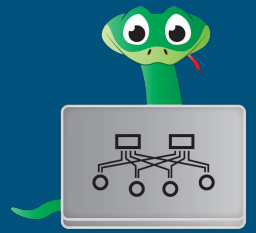
To exit

:wq - saves file and exits
:q!  - exits WITHOUT saving

u - undo the last command
yy - yank a line
p - put a line

REMEMBER:
<esc> is your friend

# What and Why: Git



PYTHON
FOR NETWORK ENGINEERS

- Why care about Git?
- Git and GitHub
- Some principles of how Git works
  - Tracking files and directories across time
  - All objects are stored in the .git directory
  - You can swap your working set of files
  - Distributed

# Git'ing Started... (we're sorry)

PYTHON
FOR NETWORK ENGINEERS

- Creating a repository on GitHub
- Cloning a repository
- git init
- Files have four different states:
  - Untracked
  - Modified
  - Staged
  - Committed

*I blame Carl for all the bad puns....*
*^ facts /CM*

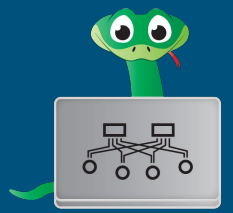
# Git Adding/Removing Files

- git status *# basically what is the current state of this repository*
- git branch *# which branches are there and which branch am I working on*
- Adding/Removing files
  - git add / git rm / git commit
  - git diff *# to see what changed on a file or set of files*
- git log *# to see the history of commits*

# Git Push & Pull

*Changes have been committed locally, but haven't been pushed up to GitHub*

- git pull / git push
  - git remote -v
  - git remote add
  - git branch -vv



PYTHON
FOR NETWORK ENGINEERS

# Git Exercises

Reference Commands:

{{ github_repo }}/day1/git/collateral/cheatsheet.md

Exercises:

{{ github_repo }}/day1/git/exercise1.md

# Git Branches

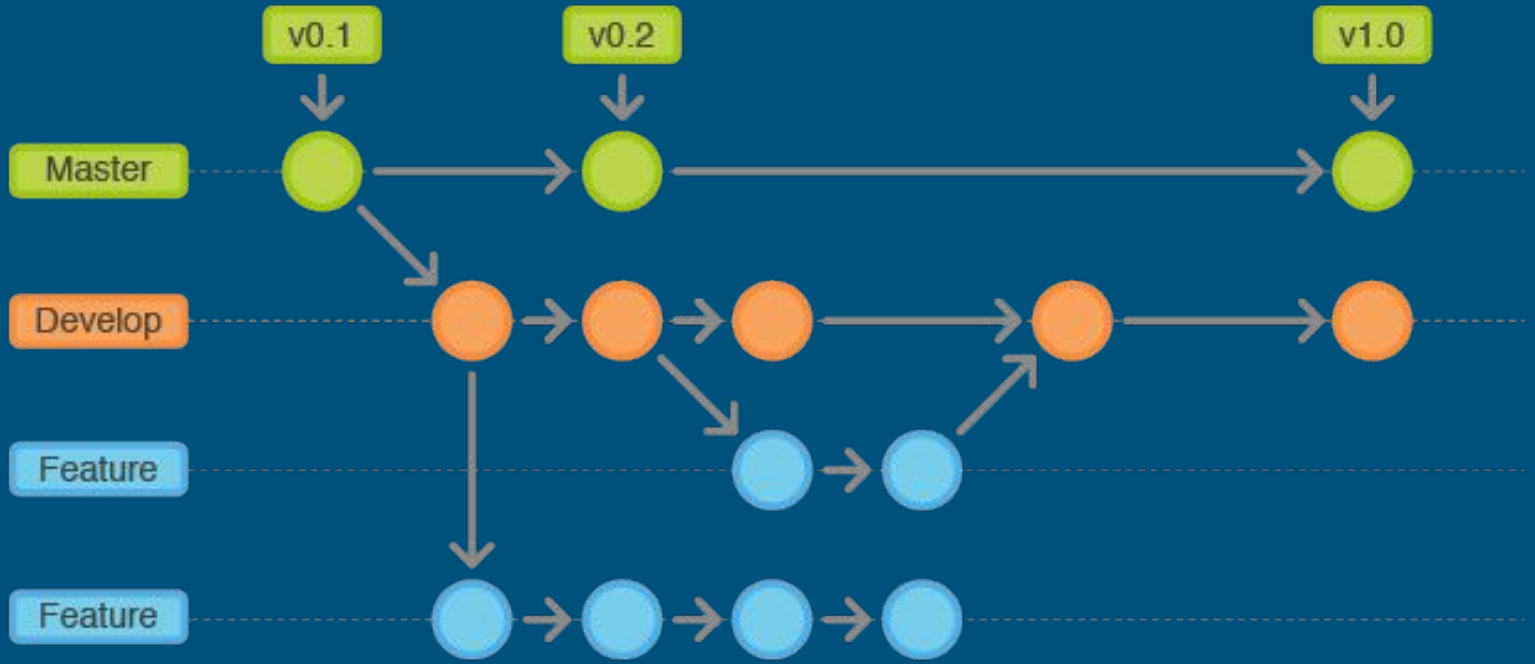
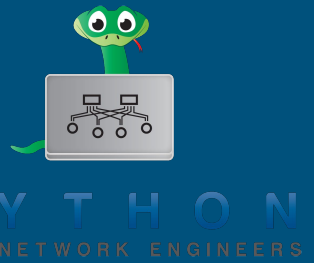



Image Source: https://leanpub.com/git-flow/read

# Git Branches




## Creating a branch

- git checkout -b dev origin/master
- git branch dev2
- git checkout dev2
- git branch *# Look at your current branches*
- Switching branches
    - Underlying files in the working directory change

## Merge operation

- Checkout the branch you want to merge into
- git merge dev2

# Git Handling Merge Conflicts

A set of changes that Git can’t reconcile

*$ git merge dev*
*Auto-merging test2.py*
*CONFLICT (content): Merge conflict in test2.py*
*Automatic merge failed; fix conflicts and then commit the result.*

$ cat test2.py

```
------------
    while True:
        print("Hello world")
        break

    for x in range(10):
        x = 0
<<<<<<< HEAD
        y = 1 * x
        z = 3
        print(y)

    print("Foo")
=======
        y += 1
        z = 3

>>>>>>> dev
```

# Git Pull Requests / Git Rebase

Pull Request - Submit changes from your copy of a repository for review and potentially integration into the main repository for the project.

Rebase - One of your branches has become out of date (relative to another copy of the repository) and you want to bring it back up to date.

PYTHON
FOR NETWORK ENGINEERS

# Git Exercises

Reference Commands:

{{ github_repo }}/day1/git/collateral/cheatsheet.md

Exercises:

{{ github_repo }}/day1/git/exercise2.md

# Why Python?

- Widely supported
  - Libraries covering all sorts of use cases
  - Runs on pretty much everything
- Language accommodates beginners through advanced.
- Promotes Maintainable / Readable Code
- Allows for easy code reuse
  - Classes, functions, modules, packages
- High-level / General Purpose Language.

# Python Characteristics

- Indentation matters!
  - Spaces, not tabs!
  - Four spaces!
- Python Programmers are very particular!
- Python 2 vs Python 3
  - Battle is over... (it has been for a while now!)
  - Python 2 support ends 1 January 2020!
- Dynamically typed
- Interpreted
- “Batteries included”

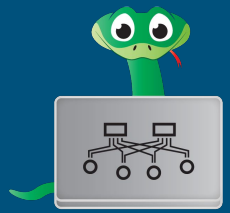
# General Items

- Python Interpreter shell (REPL)
  - Read, Evaluate, Print, Loop
- Variable assignment and naming
  - `my_var = 1234`
  - `Variable naming rules`
- Printing
  - *print(“stuff here!”)*
  - *print(my_var)*
- Creating and executing a script
  - #!/usr/bin/env python
  - chmod + x
  - python(version) myscript.py

# General Items

- Quotes, double quotes, triple quotes, oh my!
  - x = ‘this is a string’
  - x = “this is also a string”
  - x = “””a multiline string”””
- Comments
  - # comments are good!
  - “””big comments are good too!”””
- `dir()` and `help()`
  - help(some_function)
  - dir(some_function)

# Strings



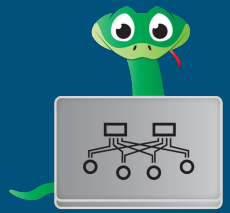
PYTHON
FOR NETWORK ENGINEERS

- String methods
  - split() — split on some delimiter (whitespace by default)
  - strip() — strip leading/trailing whitespace
  - upper() — make it all upper case!
  - Lots more!
- “in” checks for membership (for strings: substring in broader string)

```
if “str” in “substring”:
    do a thing
```

- “Chain” string methods together
  - my_string.strip().upper().split()

# Strings



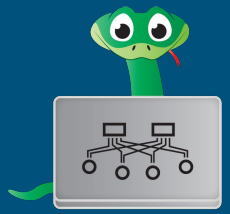
PYTHON
FOR NETWORK ENGINEERS

- Unicode (all Python3 strings are unicode)
  - print("\u2168")
- Raw strings
  - print(r"\n<- see a newline!")
- String formatting
  - .format()
    - print(“some useful info for {}”.format(person))
  - F-Strings
    - print(f”some useful info for {person}”)

# String Exercises

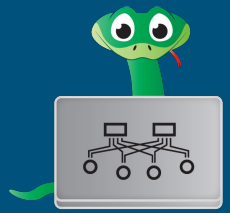
Exercises:

{{ github_repo }}/day1/strings/exercise1.md
{{ github_repo }}/day1/strings/exercise2.md



PYTHON
FOR NETWORK ENGINEERS

# Numbers



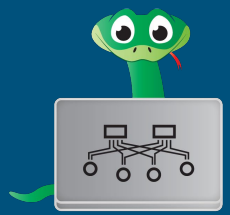
PYTHON
FOR NETWORK ENGINEERS

- Integers
  - my_int = 1234
- Floats
  - my_float = 1.234
- Operators
  - Addition (+) / Subtraction (-)
  - Multiplication (*) / Division (/)
  - Exponent (**) / Modulus (%)
    - Modulus returns remainder of the quotient ex:
      - 5 % 2 -> 1
- Note: Math operators may work in other places too!
  - print(“-” * 80) -> will print a “-” 80 times

# Numbers Exercises

Exercises:

{{ github_repo }}/day1/numbers/exercise1.md



PYTHON

FOR NETWORK ENGINEERS

# Writing to a file/reading from a file:

- Basic Method:

```
my_file = open("somefile.txt")
```

- Better Method (Context Manager):

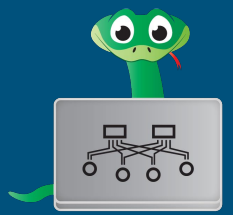
```
with open("somefile.txt") as infile:
    my_file = infile.read()
```

- File modes:
  - "r": read (default)
  - "w": write
  - "a": append

# Files Exercises

Exercises:

{{ github_repo }}/day1/files/exercise1.md



PYTHON

FOR NETWORK ENGINEERS

# Lists

- Zero-indexed!
- List methods:
  - append() — add item to end of the list
  - pop() — remove item at given index
  - sort() — sort list items
- Related String method:
  - join() — join members of iterable with a str separator. Ex:

```
>>> my_list = ["one", "two"]
>>> x = "-".join(my_list)
>>> x
"one-two"
```

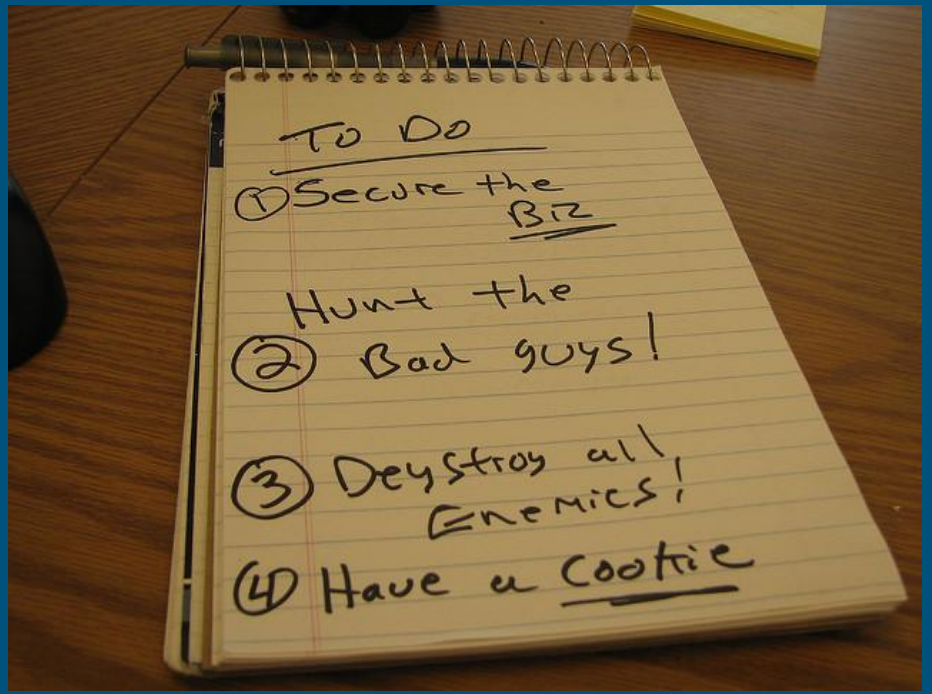



Photo: Purple Slog (Flickr)

# Lists

- Slicing
  - “Slice” a list into a new list

```
>>> my_list = ["one", "two", "three", "four", "five"]
>>> x = my_list[1:4]
>>> x
['two', 'three', 'four']
```

- Copy a list… a bit more complicated than it may seem!

```
>>> my_list = ["one", "two", "three", "four", "five"]
>>> new_list = my_list
>>> new_list.append(“six”)
>>> print(my_list)
>>> ["one", "two", "three", "four", "five", “six”]
```

# Lists

- Copy a list… for real this time… (shallow copy)

```
>>> my_list = ["one", "two", "three", "four", "five"]
>>> new_list = my_list.copy()
>>> new_list.append(“six”)
>>> print(my_list)
>>> ["one", "two", "three", "four", "five"]
>>> print(new_list)
>>> ["one", "two", "three", "four", "five", “six”]
```

- Tuple
  - Very similar to a list, but is IMMUTABLE!

# Lists

- Tuple
  - Very similar to a list, but is IMMUTABLE! (unchangeable)

```
>>> my_list = ["one", "two"]
>>> my_list.append("three")
>>> my_tuple = ("one", "two")
>>> my_tuple.append("three")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'append'
>>>
```

# Lists Exercises

Exercises:

{{ github_repo }}/day1/lists/exercise1.md

{{ github_repo }}/day1/lists/exercise2.md

# Booleans and None

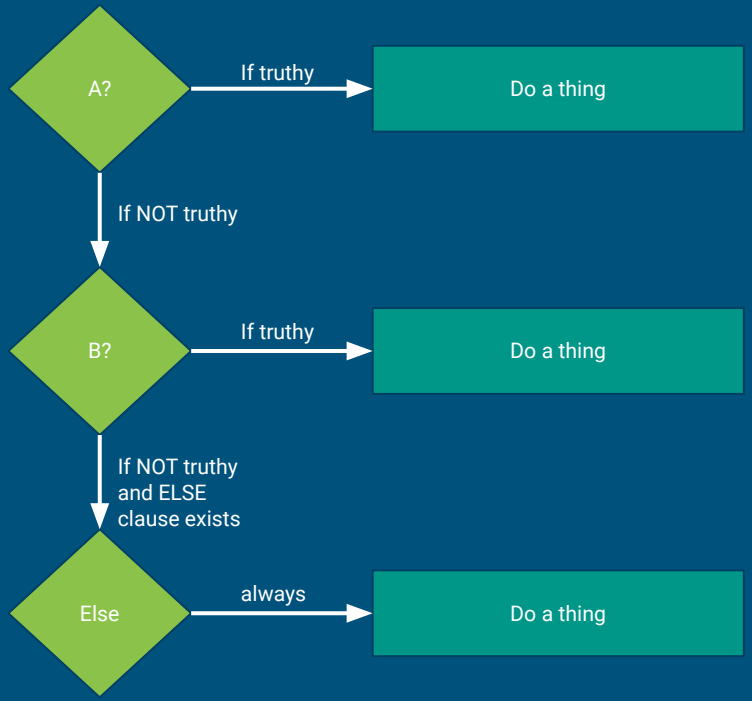
- Booleans
  - True, False
- Boolean Operators
  - and, or, not
- Identity (crisis) vs Equality
  - Identity: is, is not
  - Equality: “==”, “!=”
  - Other equality comparison operators: “<”, “>”, “<=”, “>=”
- What is TRUE anyways…
- None, the absence of… things?

```
>>> a = [1, 2, 3]
>>> b = a
>>> a == b
True
>>> a is b
True
>>> c = list(a)
>>> a == c
True
>>> a is c
False
>>> bool(a)
True
>>> bool(0)
False
>>> bool(None)
False
>>> if a is not None:
print("yay!")
...
yay!
>>>
```

# Conditionals

- Evaluate “truthiness” at each check

```
if a == 15:
    print(“Hello”)
elif a == 16:
    print(“Racecar”)
elif a == 412:
    print(“Tacocat”)
elif a >= 7:
    print(“Something”)
else:
    print(“Nada”)
```



# Loops

- “for” loop
  - Loop over a collection of things (a common case is looping over a list).
  - Similar to a “foreach” in other languages

```
>>> my_list = ["one", "two", "three"]
>>> for x in my_list:
...     print(x)
...
one
two
three
```

Image Source: Mário Monte Filho (Flickr)

# Loops

- “while” loop
  - While some condition is True, execute this code

```
>>> my_list = ["one", "two", "three"]
>>> counter = 0
>>> while counter < len(my_list):
...     print(my_list[counter])
...     counter += 1
...
one
two
three
```

# Loops

- Loop controls
  - Break: terminate “nearest enclosing loop”
  - Continue: jump to end of current iteration
- Range/Enumerate
  - Range returns a “sequence” of numbers
  - Enumerate provides both the index and the value.

```
>>> for x in range(10):
...     print(x)
...
0
<SNIP>
9
>>> my_list = ["one", "two", "three"]
>>> for index, item in
enumerate(my_list):
...     print(index, item)
...
0 one
1 two
2 three
>>>
```

# Loops Exercises

Exercises:

{{ github_repo }}/day1/loops/exercise1.md
{{ github_repo }}/day1/loops/exercise2.md
{{ github_repo }}/day1/loops/exercise3.md

# Dictionaries

- Structured key/value store
- Keys are almost always strings
- Values can be any Python object
- Historically unordered; ordered in newer Python

```
>>> my_dict = {}
>>> my_dict["akey"] = "avalue"
>>> my_dict
{'akey': 'avalue'}
>>> my_dict["akey"] = "anewvalue"
>>> my_dict
{'akey': 'anewvalue'}
>>> my_dict[1] = "intaskey"
>>> my_dict
{'akey': 'anewvalue', 1: 'intaskey'}
>>> del(my_dict[1])
>>> my_dict
{'akey': 'anewvalue'}
>>>
```

# Dictionaries

- Dictionary methods
  - get()
  - pop()
  - keys()
  - values()
  - items()
- Copying has similar challenges as lists!
  - copy.deepcopy() copies nested values in a dictionary!

Image Source: Holger Zscheyge (Flickr)

# Dictionaries

- get() “gets” value of a given key
- Does not fail if key is not present (returns None)
- Optionally assign a default value if key not present

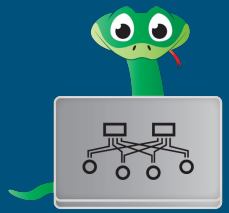
```
>>> my_dict
{'akey': 'anewvalue', 'racecar': 'ferrari', 'meme': 'tacocat'}
>>> x = my_dict.get("racecar")
>>> x
'ferrari'
>>> x = my_dict.get("notakey")
>>> x
>>> x = my_dict.get("notakey", "Nothing to see here")
>>> x
'Nothing to see here'
```

# Dictionaries

- pop() “removes” and assigns value of a given key
- Returns KeyError if key is not present
- Optionally assign a default value if key not present

```
>>> my_dict
{'akey': 'anewvalue', 'racecar': 'ferrari', 'meme': 'tacocat'}
>>> x = my_dict.pop("racecar")
>>> x
'ferrari'
>>> x = my_dict.pop("notakey", "Nope!")
>>> x
'Nope!'
>>>
```

# Dictionaries



PYTHON
FOR NETWORK ENGINEERS

- keys() returns “list” of keys
- values() returns “list” of values
- items() returns “list” of tuples of keys/values

```
>>> my_dict.keys()
dict_keys(['akey', 'meme'])
>>> my_dict.values()
dict_values(['anewvalue', 'tacocat'])
>>> my_dict.items()
dict_items([('akey', 'anewvalue'), ('meme', 'tacocat')])
>>>
```

*Technically not a “list”, but a list-like object*

# Dictionaries Exercises

Exercises:

{{ github_repo }}/day1/dictionaries/exercise1.md

PYTHON

FOR NETWORK ENGINEERS

# Exception Handling

- “Try” to gracefully handle errors
- “Except” exception that would be raised
- Optional “Finally” do something else (always happens)

```
>>> try:
...     my_dict["tacos"]
... except KeyError:
...     print("Sad... no tacos :(")
...
Sad... no tacos :(
```

# Exception Handling Exercises

Exercises:

{{ github_repo }}/day1/exception/exercise1.md

# Day 1 Complete!

# Day 2 Schedule

- Review!

- Regular Expressions
- Reusable code with Functions

- Netmiko
- More Netmiko

- More Reusable code - Classes/Objects
- Python Code Structure


Image Source: au_tiger01 (Flickr)

# Day 1 Review

- Git
  - git [add|commit|push|pull|status]
- Strings/Numbers
  - String formatting / dealing with types — i.e. int vs float vs str
  - Math on strings! Concatenation of str/int
- Files
  - Context Managers are good!
- Lists
  - Zero indexed, creating, accessing, modifying
- Loops
  - for/while

# Day 1 Review

- Dictionaries
  - Key/Value pair
  - Ordered, but order shouldn't matter as the "keys" are the index!
- Booleans/Conditionals
  - == vs "is"
  - Truthiness
- Exceptions
  - *Try* to catch exact exceptions where possible
  - .get() can help avoiding common KeyError exception/issue

# Day 1 Review Exercises

Exercises:

{{ github_repo }}/day1/recap/exercise1.md

{{ github_repo }}/day1/recap/exercise2.md

PYTHON

FOR NETWORK ENGINEERS

# Regular Expressions

- Super powerful “find and replace”!
- Cryptic patterns to ensure nobody can read the regex you write (including yourself 3 weeks from now)
- “re” module (standard library)
  - search (given pattern anywhere in string)
  - findall (of a given pattern in string)
  - sub (substitute pattern in string)


Image Source: XKCD https://xkcd.com/208/

# Regular Expressions - Pattern Basics

| Character Classes | Quantifiers | Flags |
| --- | --- | --- |
| Symbol: “\w”, Name: “word”<br>match any alphanumeric & underscore “word” | Symbol: “+”, Name: “plus”<br>match one or more of preceding token | Flag: “re.M”, Name: “multiline”<br>Anchors (^,$) match on each line instead of the entire string. |
| Symbol: “\d”, Name: “digit”<br>match any digit character 0-9 | Symbol: “*”, Name: “star”<br>match zero or more of preceding token | Flag: “re.I”, Name: “ignore case”<br>make all matches case insensitive. |
| Symbol: “\s”, Name: “whitespace”<br>match any whitespace character (space, tabs, line breaks) | Symbol: “{1,5}”, Name: “quantifier”<br>match X up to Y ({X,Y}) of previous token | Flag: “re.DOTALL”, Name: “dotall” Make period character also match newlines. By default, “.*” won’t span across newlines. |
| Symbol: “.” Name: “dot”<br>match any character except line breaks | | |

# Regular Expressions

- Always use “raw” strings
- Parenthesis used to retain a pattern (group)
- Name patterns: (?P<software_ver>Ver.*)
- Greedy vs Lazy (default greedy):
  - “?” is lazy quantifier

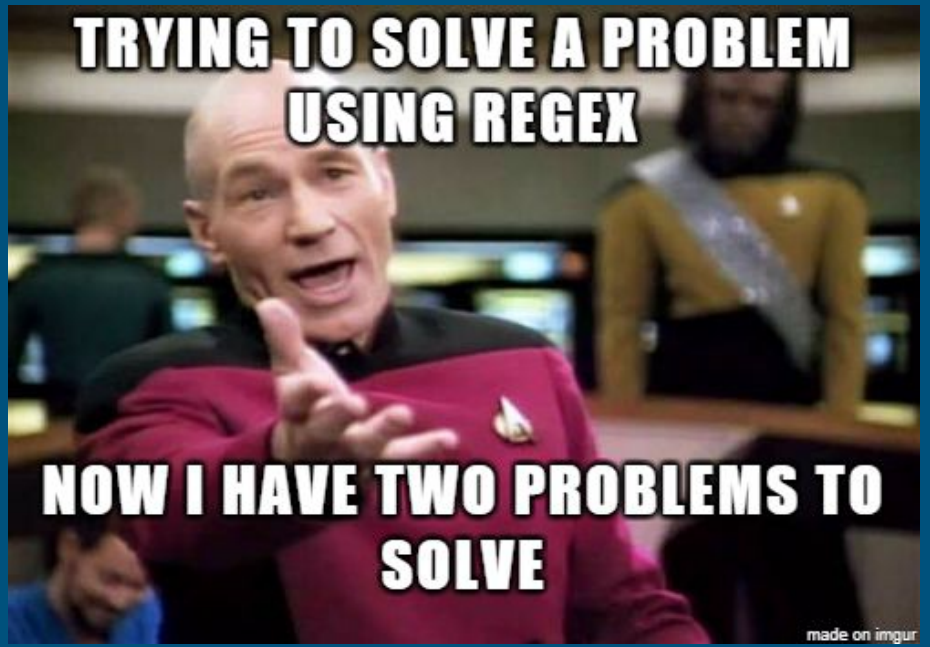



Image Source: SirDoItAlot

# Regular Expressions - Examples

- “search” - Return first pattern match from a string:

```
>>> sh_version = """Cisco IOS Software, C3560CX Software
SNIP
cisco WS-C3560CX-8PC-S (APM86XXX) processor (revision A0) with
>>> res = re.search(r"ws-c3560[\w\d-]*", sh_version, flags=re.I)
>>> res
<re.Match object; span=(1498, 1514), match='WS-C3560CX-8PC-S'>
>>> res.group()
'WS-C3560CX-8PC-S'
>>>
```

- Flags can modify behavior of matching (multiline, case insensitive, dotall)

# Regular Expressions - Examples

- “findall” - Return all hostnames from a string:

```
>>> hosts = "cisco1.lasthop.io, arista1.lasthop.io"
>>> res = re.findall(r"\w+\.\w+\.\w+", hosts)
>>> res
['cisco1.lasthop.io', 'arista1.lasthop.io']
```

- “sub” - Replace “:80” with “:443” in a given string:

```
>>> ahost = "somehost:80"
>>> res = re.sub(r":80", ":443", ahost)
>>> res
'somehost:443'
```

# Regular Expressions - Resources

- Intro Tutorial
  - https://regexone.com/lesson/introduction_abcs
- Regex Online Testers
  - https://regex101.com
  - https://pythex.org
- Python re module docs
  - https://docs.python.org/3/library/re.html

# Regular Expressions Exercises

Exercises:

{{ github_repo }}/day2/regex/exercise1.md

{{ github_repo }}/day2/regex/exercise2.md

PYTHON

FOR NETWORK ENGINEERS

# Functions

- What even is a function?
  - A chunk of code that can be “called” (told to execute)
  - Yes, “print” is a function! So is “dir”.

- Why care?
  - Functions are reusable!
  - DRY*: do not repeat yourself
  - Single Responsibility Principle*: module/function/class should do just *one* thing

```
>>> def hello_world():
...     print("Hello, world!")
...
>>> hello_world()
Hello, world!
>>> hello_world()
Hello, world!
>>> hello_world()
Hello, world!
>>> hello_world()
Hello, world!
>>> hello_world()
Hello, world!
>>> hello_world()
Hello, world!
```

# Functions

- Positional Arguments
  - As the name implies are based on position

```
>>> def hello_world_v2(arg1, arg2, arg3):
...     print(f"Hello, {arg1}, {arg2}, {arg3}!")
...
>>> hello_world_v2("ollie", "luna","luca")
Hello, ollie, luna, luca!
>>> hello_world_v2("luca", "luna", "ollie")
Hello, luca, luna, ollie!
>>>
```

# Functions

- Keyword Arguments
  - Order does not matter because, like a dictionary, keys are used to associate arguments.

```
>>> def hello_world_v2(arg1, arg2, arg3):
...     print(f"Hello, {arg1}, {arg2}, {arg3}!")
...
>>> hello_world_v2("ollie", "luna", arg3="luca")
Hello, ollie, luna, luca!
>>> hello_world_v2("ollie", arg3="luna", arg2="luca")
Hello, ollie, luca, luna!
>>>
```

# Functions

- Default Parameters
  - Function parameters with default values.
  - Useful for parameters that *usually* are set to one value but can sometimes be overridden.
  - Allows you to augment existing functions.

```
>>> def hello_world_v2(arg1, arg2, arg3="seattle"):
...     print(f"Hello, {arg1}, {arg2}, {arg3}!")
...
>>> hello_world_v2("san francisco", "portland")
Hello, san francisco, portland, seattle!
>>> hello_world_v2("san francisco", "portland", "vancouver")
Hello, san francisco, portland, vancouver!
>>>
```

# Functions

- Argument “unpacking”
  - Unpack an iterable with “*” (convert a list from being one argument to being an argument for each list element)

```
>>> def hello_world_v3(arg1, arg2, arg3):
...     print(f"Hello, {arg1}, {arg2}, {arg3}!")
...
>>> list_of_args = ["sfo", "pdx", "sea"]
>>> hello_world_v3(*list_of_args)
Hello, sfo, pdx, sea!
>>>
```

# Functions

- Keyword Argument “unpacking”
  - Unpack a dict with “**” — convert dictionary from being one argument to individual key-value pairs (for arguments)

```
>>> def hello_world_v3(arg1, arg2, arg3="seattle"):
...     print(f"Hello, {arg1}, {arg2}, {arg3}!")
...
>>> dict_of_args = {"arg1": "sfo", "arg2": "pdx", "arg3": "sea"}
>>> hello_world_v3(**dict_of_args)
Hello, sfo, pdx, sea!
>>> dict_of_args.pop("arg1")
'sfo'
>>> small_list = ["sfo"]
>>> hello_world_v3(*small_list, **dict_of_args)
Hello, sfo, pdx, sea!
```

# Functions Exercises

Exercises:

{{ github_repo }}/day2/functions/exercise1.md
{{ github_repo }}/day2/functions/exercise2.md
{{ github_repo }}/day2/functions/exercise3.md
{{ github_repo }}/day2/functions/exercise4.md

# Netmiko Introduction


- “Multi-vendor library to simplify Paramiko SSH connections to network devices”
- Lower-level // (p)expect → Paramiko → Netmiko // Higher-level
- Paramiko is the “standard” SSH Python library
- Netmiko abstracts paramiko and makes it network device-friendly
- Netmiko: https://github.com/ktbyers/netmiko

# Netmiko Platforms/Vendors


- Currently (very) roughly 60 different platforms supported by Netmiko.
- Three different categories of supported platform:
  - Regularly tested
  - Limited testing
  - Experimental
- Platforms: https://ktbyers.github.io/netmiko/PLATFORMS.html

| **Regularly Tested Platforms** | | | |
|---|---|---|---|
| Arista EOS | Cisco ASA | Cisco IOS | Cisco IOS-XE |
| Cisco IOS-XR | Cisco NX-OS | Cisco SG300 | HP ProCurve |
| Juniper Junos | Linux | | |

# Netmiko ConnectHandler


- “ConnetHandler” is entry point into Netmiko
- Accepts many arguments to establish SSH (or telnet) connection such as:
  - ip/host
  - port
  - username
  - password
  - SSH Key Information (key_file)
  - etc.
- Determines proper device handling based on “device_type” argument
- Common to invoke ConnectHandler using **device

# Netmiko Example


```
>>> from getpass import getpass
>>> from netmiko import ConnectHandler
>>>
>>> my_device = {}
>>> my_device["device_type"] = "arista_eos"
>>> my_device["host"] = "arista1.lasthop.io"
>>> my_device["username"] = "pyclass"
>>> my_device["password"] = getpass()
Password:

>>> conn = ConnectHandler(**my_device)

>>> print(conn.find_prompt())
arista1#
>>>
```

# Netmiko - Key Methods


| Method | Purpose |
|---|---|
| send_command() | Send command, use pattern matching to know when “done” |
| send_command_timing() | Send command, use timing to know when “done” |
| send_config_set() | Send list of configuration commands |
| send_config_from_file() | Send configuration commands from a file |
| save_config() | Delete config... just kidding… save the config |
| commit() | Commit configuration (for specific platforms) |
| enable() | Enter “enable”/privilege mode |
| disconnect() | Close connection |
| write_channel() | Write to channel directly (bypass Netmiko prompt searching/timing |
| read_channel() | Read directly from channel (bypass Netmiko prompt searching/timing) |
| File Transfer Class | SCP files to/from devices |

# Netmiko Session Log


```
SNIP
>>> my_device["session_log"] = "session.log"
Password:

>>> conn = ConnectHandler(**my_device)

>>> print(conn.find_prompt())
arista1#

$ cat session.log
SNIP
arista1#terminal length 0
Pagination disabled.
arista1#terminal width 511
Width set to 511 columns.
arista1#
arista1#
```

# Netmiko Exercises


Examples:

https://github.com/ktbyers/netmiko/tree/develop/examples/use_cases

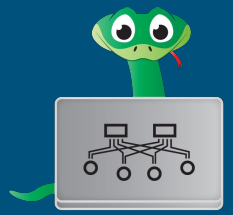
Reference Material:

{{ github_repo }}/day2/netmiko/collateral/

Exercises:

{{ github_repo }}/day2/netmiko/exercise1.md
{{ github_repo }}/day2/netmiko/exercise2.md

# Classes and Objects



PYTHON
FOR NETWORK ENGINEERS

- What even is a Class?
  - A “template” for creating objects
  - Can have associated methods (functions of a class) and attributes

```
>>> class ClassA():
...     def __init__(self, name):
...         self.name = name
...     def say_hello(self):
...        print(f"{self.name} says hello!")
```

# Classes and Objects

- What even is an Object??
  - An instance of a class — in other words an individual representation of the class.

```
>>> class ClassA():
...     def __init__(self, name):
...         self.name = name
...     def say_hello(self):
...        print(f"{self.name} says hello!")

>>> carl_obj = ClassA("carl")
>>> carl_obj.say_hello()
carl says hello!
```

# Classes and Objects

PYTHON
FOR NETWORK ENGINEERS

- Why care?
  - Classes are reusable!
  - DRY*: do not repeat yourself
  - Single Responsibility Principle*: module/function/class should do one *one* thing

```
>>> class ClassA():
...     def __init__(self, name):
...         self.name = name
...     def say_hello(self):
...        print(f"{self.name} says hello!")
>>> carl_obj = ClassA("carl")
>>> carl_obj.say_hello()
carl says hello!
>>> kirk_obj = ClassA("kirk")
>>> kirk_obj.say_hello()
kirk says hello!
```

# Classes

- Classes generally have a special “__init__” method that is called as part of the object creation process.
- Functions that are part of a class are called “methods”
  - “methods” usually* contain a first argument named “self”
  - “self” is a reference to the object itself
- Inheritance is mostly* out of scope for now, but can be powerful/useful for reducing code duplication (i.e. for sharing common code).

# Classes and Objects Exercises

Exercises:

{{ github_repo }}/day2/classes/exercise1.md

{{ github_repo }}/day2/classes/exercise2.md

PYTHON

FOR NETWORK ENGINEERS

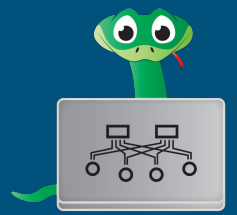
# Python Code Structure

- Imports at top of the file
- Naming Standards
  - CONSTANT variables in all upper
  - Variables and functions—all lower case with “_” word separation
  - Classes use “CapWords”
- if __name__ == “__main__”:
- main() function call

# Structure Exercises

Exercises:

{{ github_repo }}/day2/structure/exercise1.md
{{ github_repo }}/day2/structure/exercise2.md



PYTHON
FOR NETWORK ENGINEERS

# Day 2 Complete!

# Day 3 Schedule

- Review!
- Modules/Packages
- sys.path and PYTHONPATH
- Libraries/PIP/Venv/Linting
- TextFSM/Genie (optional)
- pdb - Python debugger
- Complex Data Structures
- Cisco NX-API
- Requests and RESTful APIs


Image Source: Flickr: Pierre-Olivier Carles

# Day 2 Review

PYTHON
FOR NETWORK ENGINEERS

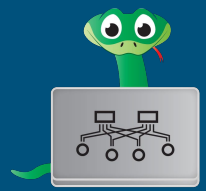
- Regex
  - We still frequently encounter large-string blocks that we must parse.
- Functions
  - DRY, Single Responsibility Principle — write reusable code!
- Netmiko
  - Simplify legacy device management using Python (SSH/telnet)
- Classes and Objects
  - OOP — class is a “template”; object is an instance of that template
- Code Structure
  - OCD is not all bad! Standards/style make your life easier! (especially across team members)

# Day 2 Review Exercises

Exercises:

{{ github_repo }}/day2/recap/exercise1.md

{{ github_repo }}/day2/recap/exercise2.md



PYTHON

FOR NETWORK ENGINEERS

# Modules, Packages, Imports

- Module
  - A Python file that you can import into another Python program
- Python Package
  - An importable Python **directory**
  - __init__.py indicates a **package**
- Imports
  - import my_library -> library now in your namespace (for example: re.search)
  - from my_library import resource -> import a specific resource into your namespace

# How does Python find things? sys.path and PYTHONPATH

- “sys.path” is a list of places Python will look to find modules/packages

```
$ python -c "import sys; print(sys.path)"
['', '/usr/lib64/python3.6',
'/usr/lib64/python3.6/lib-dynload', <SNIP>
'/home/user/VENV/py3_venv/lib/python3.6/site-packages']
```

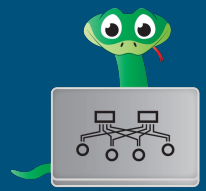
- PYTHONPATH is an environment variable that can modify sys.path

```
$ export PYTHONPATH=/home/user/python-libs
$ python -c "import sys; print(sys.path)"
['', '/home/user/python-libs', '/usr/lib64/python3.6',
'/usr/lib64/python3.6/lib-dynload', <SNIP>
'/home/user/VENV/py3_venv/lib/python3.6/site-packages']
```

# Modules, Packages, and Paths Exercises

Exercises:

{{ github_repo }}/day3/modules/exercise1.md
{{ github_repo }}/day3/modules/exercise2.md



PYTHON
FOR NETWORK ENGINEERS

# PIP and Libraries

- PIP —> PIP Installs Packages
- PyPi —> Python Package Index
- pip list
- pip search
- pip install
- pip install netmiko==2.4.2
- pip install -e .
- pip install -r requirements.txt
- pip freeze

“python -m pip” - can be your friend

Image Source: Viva Vivanista (Flickr)

# Virtual Environments

- VRF for Python packages (kinda)!
- Avoid inconsistent requirements
- Avoid stomping on sys packages
- python -m venv [NAME]
- source [NAME]/bin/activate
- deactivate

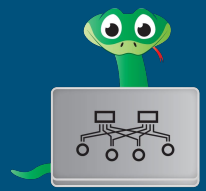

Image Source: truthexists
(Imageflip)

# Virtual Environment Exercises

Exercises:

{{ github_repo }}/day3/virtualenv/exercise1.md

{{ github_repo }}/day3/virtualenv/exercise2.md



PYTHON

FOR NETWORK ENGINEERS

# Linting

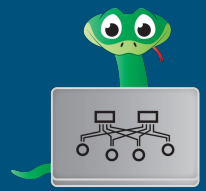

- PEP8 (Python Enhancement Proposal) == defacto style guide
- pylint, pycodestyle, pylama good linter choices (pylint is a bit verbose).
- Consistency and conventions make life easier.
- Opinionated auto-formatting is the new craze! Python “black”
- Standards/PEP are good, but don’t let it trick you into writing bad code!

# Linting Exercises

Exercises:

{{ github_repo }}/day3/linting/exercise1.md

{{ github_repo }}/day3/linting/exercise2.md



PYTHON

FOR NETWORK ENGINEERS

# *The Network Device Parsing Problem*

Avoid the problem:

- Use good APIs
- Use “| json” or “| xml”

Get someone else to do the parsing for you:

- ntc-templates
- Cisco genie
- NAPALM getters

Parse yourself...regular expressions or your own TextFSM templates.

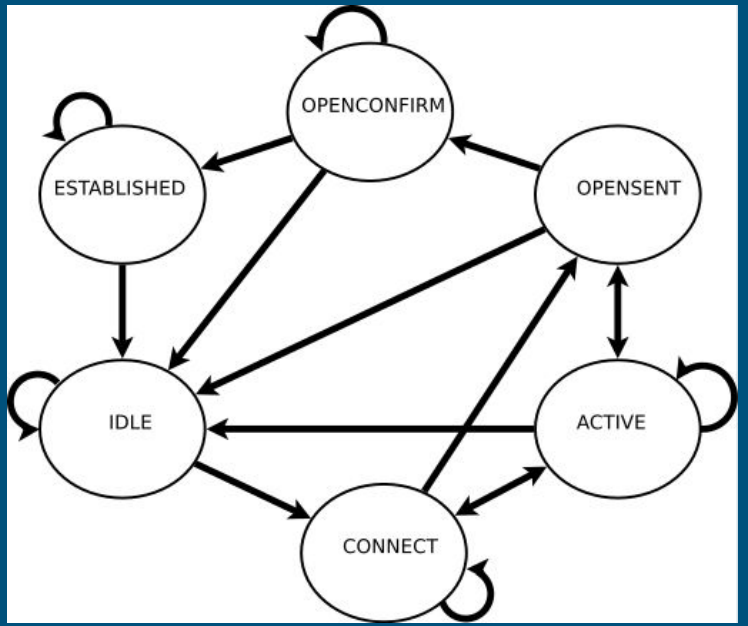
# TextFSM

Text + Finite State Machine

TextFSM is a library originally created by Google for parsing complex unstructured strings.

*Because who doesn't want to couple Regular Expressions and a State Machine.*

BGP State Machine




Source: https://commons.wikimedia.org/wiki/File:BGP_FSM_3.svg

# TextFSM

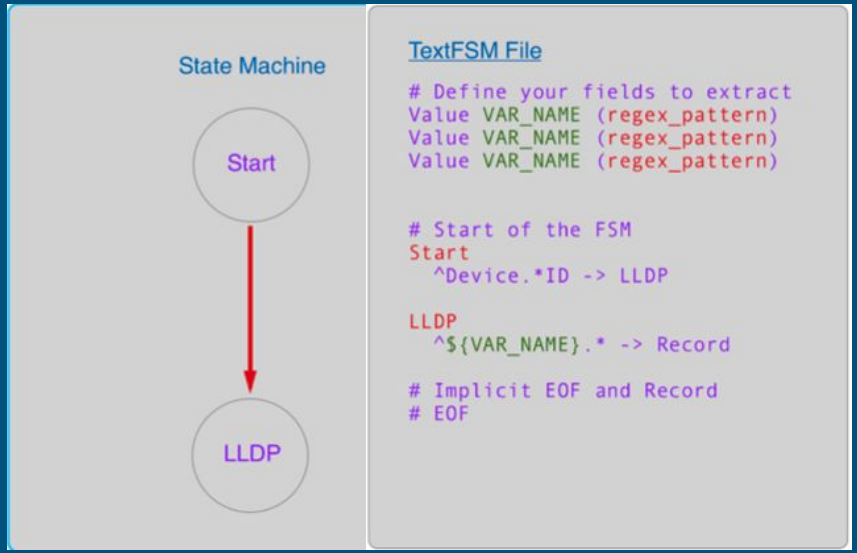
- Unstructured data (text) is messy!

```
$ 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

- Variables > Start > State Transition
- Implicit (or explicit) EOF
- pip install textfsm
- NTC Templates for many network platforms/commands
  - pip install ntc-templates
  - Or clone repo
- But wait… there's more!



# Netmiko and TextFSM

```
$ python netmiko_textfsm.py

Protocol  Address          Age (min)  Hardware Addr   Type   Interface
Internet  172.31.255.255          -  c800.84b2.e9c4  ARPA   Vlan3967
Internet  172.31.255.254        134  8478.acae.c196  ARPA   Vlan3967

[{'protocol': 'Internet', 'address': '172.31.255.255', 'age': '-',
'mac': 'c800.84b2.e9c4', 'type': 'ARPA', 'interface': 'Vlan3967'},
{'protocol': 'Internet', 'address': '172.31.255.254', 'age': '134',
'mac': '8478.acae.c196', 'type': 'ARPA', 'interface': 'Vlan3967'}]
```

# Genie

- Unstructured data (text) is messy! (again… err… still...)

```
$ 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
```

# Genie

- Genie is part of the pyats testing/automation framework.
- Let Cisco do the work of writing the parsers for you!
- pubhub.devnetcloud.com/media/genie-feature-browser/docs/#/parsers

```
>>> from netmiko import ConnectHandler
>>> conn = ConnectHandler(host='172.31.254.1', device_type='cisco_ios',
username='user', password='password')
>>> print(conn.send_command("show version", use_genie=True))
{'version': {'version_short': '15.2', 'platform': 'C3560CX', 'version':
'15.2(4)E7', 'image_id': 'C3560CX-UNIVERSALK9-M', 'os': <SNIP>
'curr_config_register': '0xF'}}
```

# TextFSM/Genie Exercises

Reference Examples:

{{ github_repo }}/day3/parsers/collateral/

Exercises:

{{ github_repo }}/day3/parsers/exercise1.md
{{ github_repo }}/day3/parsers/exercise2.md
{{ github_repo }}/day3/parsers/exercise3.md
{{ github_repo }}/day3/parsers/exercise4.md

# Data Serialization: YAML and JSON

- Why do we need serialization?
  - Transfer programming objects between different machines, different languages.
  - Often JSON/YAML files are used for storing configurations/settings.

- JSON
  - A string representation of lists, dictionaries, other data types.
  - Very similar to a Python data structure representation, but not *exactly* the same!

- YAML
  - Superset of JSON that is more human readable*

# Data Serialization Exercises

Reference Examples:

{{ github_repo }}/day3/serialization/collateral/

Exercises:

{{ github_repo }}/day3/serialization/exercise1.md
{{ github_repo }}/day3/serialization/exercise2.md
{{ github_repo }}/day3/serialization/exercise3.md

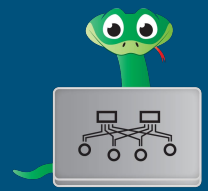
# Python Debugger

- Because you will 100% need to debug things at some point!
- (i)pdb commands:
  - (n)ext *# step one line at a time; do not “descend”*
  - (s)tep *# step one line at a time; *do* “descend”*
  - (c)ontinue *# continue execution (until completion or next breakpoint)*
  - (b)reak N *# set a breakpoint at line N*
  - (l)ist [N, N2] *# list lines at current position or starting at N, optionally through N2*
  - (d)own *# move *down* the stack*
  - (u)p *# move *up* the stack*
  - (p)rint *# print a variable*
  - *(pp)rint # pretty print a variable*
  - *(q)uit # quit pdb*

# Python Debugger Exercises

Exercises:

{{ github_repo }}/day3/debugger/exercise1.md



PYTHON

FOR NETWORK ENGINEERS

# Complex Data Structures

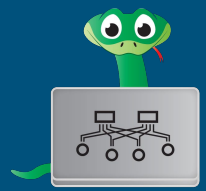
1. Investigate layer by layer (simplify the problem)
2. Determine object type (list, dict, or ?)
3. Single or multiple elements?

```
>>> indata
[{'protocol': 'O', 'type': 'E2', 'network': '0.0.0.0', 'mask': '0', 'distance': '110', 'metric': '1', 'nexthop_ip': '
    172.31.255.254', 'nexthop_if': 'Vlan3967', 'uptime': '3w6d'}, {'protocol': 'C', 'type': '', 'network': '
    172.31.254.0', 'mask': '24', 'distance': '', 'metric': '', 'nexthop_ip': '', 'nexthop_if': 'Vlan254', 'uptime': ''}
    , {'protocol': 'L', 'type': '', 'network': '172.31.254.2', 'mask': '32', 'distance': '', 'metric': '', 'nexthop_ip'
    : '', 'nexthop_if': 'Vlan254', 'uptime': ''}, {'protocol': 'C', 'type': '', 'network': '172.31.255.5', 'mask': '32'
    , 'distance': '', 'metric': '', 'nexthop_ip': '', 'nexthop_if': 'Loopback0', 'uptime': ''}, {'protocol': 'C', 'type
    ': '', 'network': '172.31.255.254', 'mask': '31', 'distance': '', 'metric': '', 'nexthop_ip': '', 'nexthop_if': '
    Vlan3967', 'uptime': ''}, {'protocol': 'L', 'type': '', 'network': '172.31.255.255', 'mask': '32', 'distance': '',
    'metric': '', 'nexthop_ip': '', 'nexthop_if': 'Vlan3967', 'uptime': ''}]
>>> type(indata)
<class 'list'>
>>> len(indata)
6
>>> indata[0]
{'protocol': 'O', 'type': 'E2', 'network': '0.0.0.0', 'mask': '0', 'distance': '110', 'metric': '1', 'nexthop_ip': '
    172.31.255.254', 'nexthop_if': 'Vlan3967', 'uptime': '3w6d'}
>>> type(indata[0])
<class 'dict'>
>>> indata[0].keys()
dict_keys(['protocol', 'type', 'network', 'mask', 'distance', 'metric', 'nexthop_ip', 'nexthop_if', 'uptime'])
```

# Complex Data Structures Exercises

Exercises:

{{ github_repo }}/day3/data_struct/exercise1.md

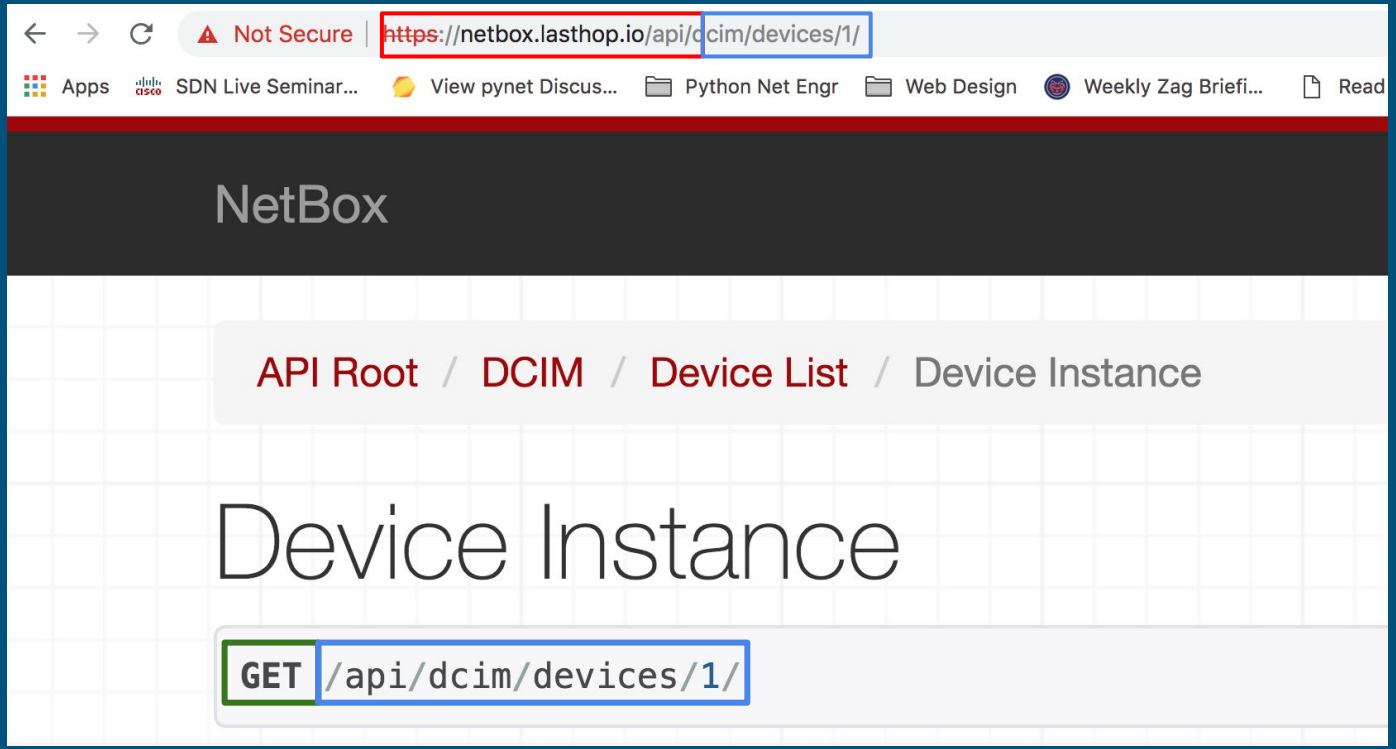


PYTHON

FOR NETWORK ENGINEERS

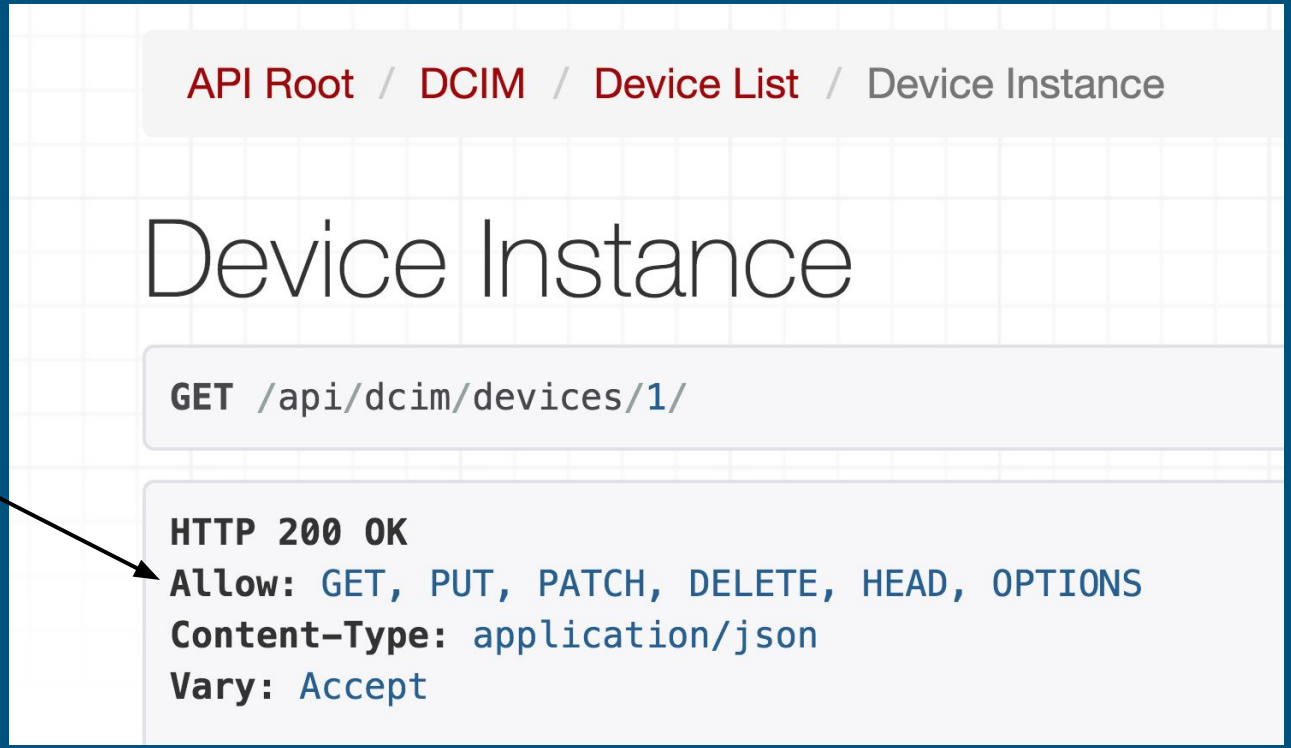
# REST API - Characteristics

- URL
- METHOD



# REST API - HTTP Methods

Available HTTP Methods



# 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 - Base GET Request (Requests)

```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 (cookie) Based
- OAuth2

```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()
```

# REST API - Recap/Strategy

1. Determine if there is an existing Python library available.
   a. If library exists, does it do everything you need? (if not: requests)
   b. If no library: requests

2. Determine how to accomplish authentication.
   a. Library should abstract much of this for you.
   b. Token based auth is common for non public web services
   c. OAuth and token auth are common for public web services

3. Gain an understanding of how to access, create, update, delete things in the API.

4. Start building up abstractions to accomplish your goals.

# API Exercises

Reference Examples:

{{ github_repo }}/day3/api/collateral/

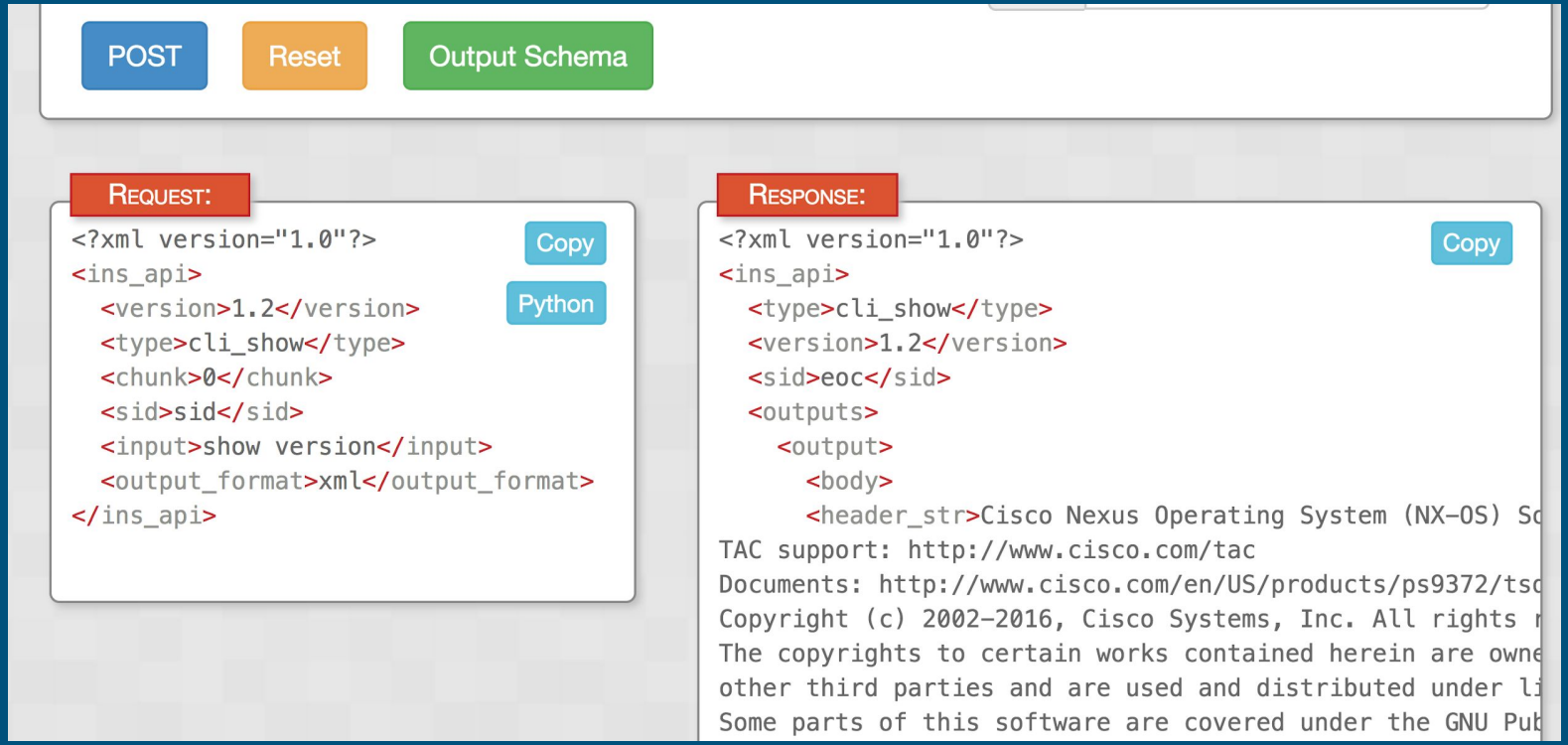
Exercises:

{{ github_repo }}/day3/api/exercise1.md
{{ github_repo }}/day3/api/exercise2.md
{{ github_repo }}/day3/api/exercise3.md
{{ github_repo }}/day3/api/exercise4.md

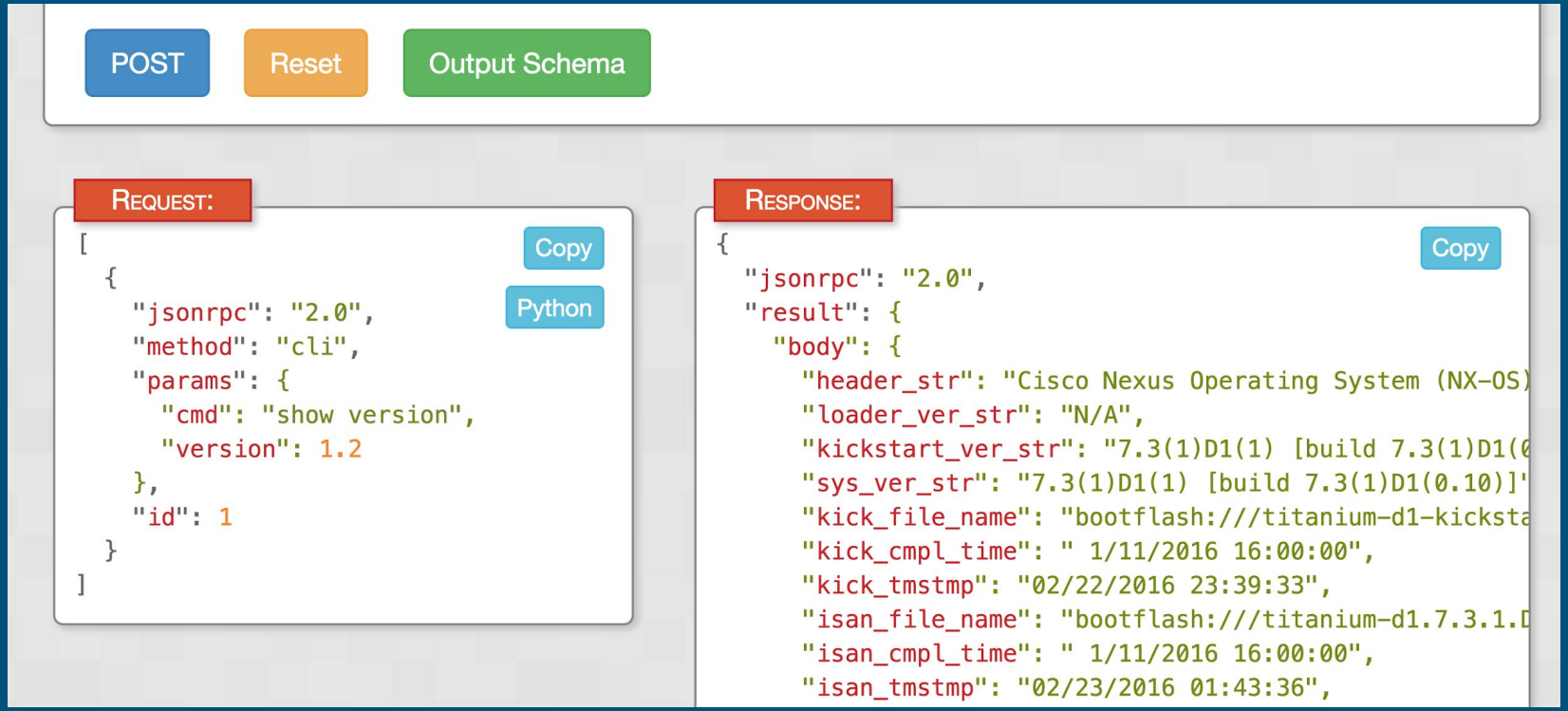
# Cisco NX-OS and NX-API

- Uses HTTP/HTTPS transport
- XML or JSON-RPC payload
- Python Libraries: nxapi-plumbing and pynxos
- NAPALM

# Cisco NX-OS and NX-API



# Cisco NX-OS and NX-API (JSON-RPC)



# Cisco NX-OS and NX-API

```
import requests
from requests.packages.urllib3.exceptions import InsecureRequestWarning
from pprint import pprint
from nxapi_plumbing import Device

requests.packages.urllib3.disable_warnings(InsecureRequestWarning)

device = Device(
    api_format="jsonrpc",| “xml”
    host="nxos1.lasthop.io",
    username="admin",
    password="password",
    transport="https",
    port=8443,
    verify=False,
)

output = device.show("show hostname")
print(output)
```

# NX-API Plumbing - Key Methods

| Method | Purpose |
| --- | --- |
| show() | Send a non-configuration command |
| show_list() | Send a list of non-configuration commands |
| config() | Send a configuration command |
| config_list() | Send a list of configuration commands |
| save() | Save running config (by default to startup-config, but configurable) |
| rollback() | Write erase, reload…? Actually this is probably rollback to a checkpoint file... |
| checkpoint() | Create a checkpoint file from the running config |

# NX-OS NX-API Exercises

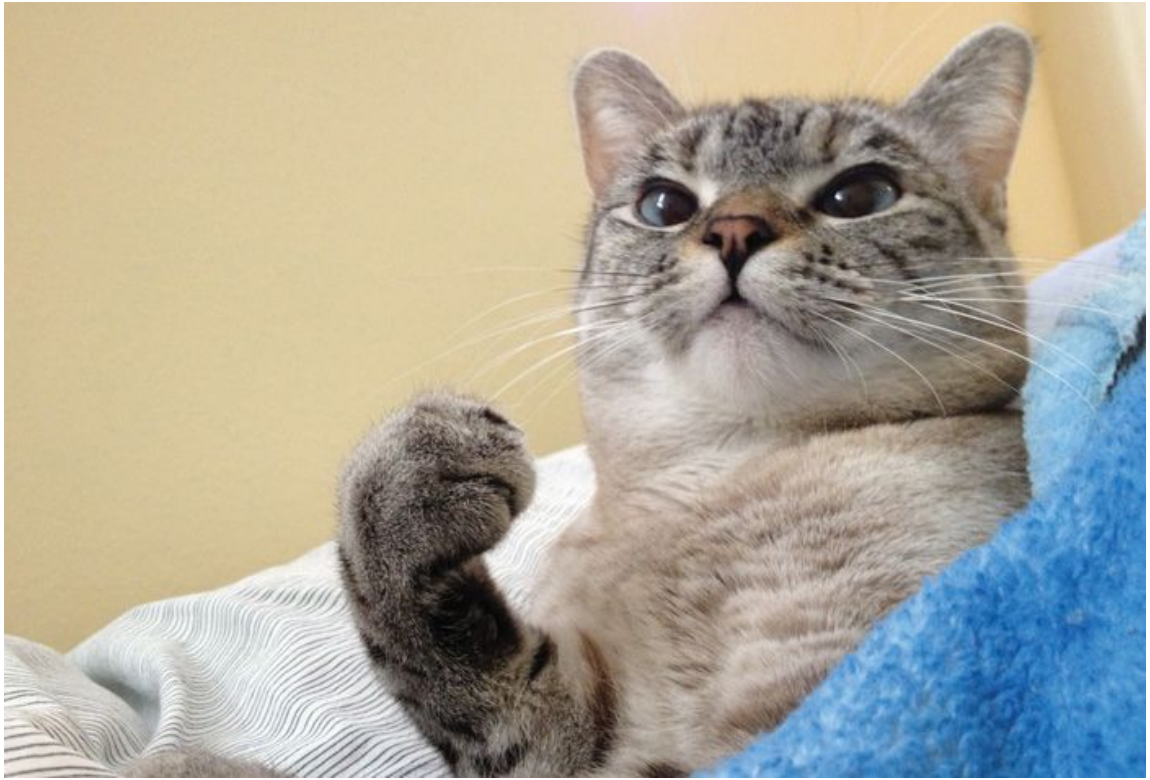
Reference Examples:

{{ github_repo }}/day3/nxapi/collateral/

Exercises:

{{ github_repo }}/day3/nxapi/exercise1.md
{{ github_repo }}/day3/nxapi/exercise2.md

# Day 3 Complete!



# Day 4 Schedule

- Review!
- Subprocess—integrating to the OS
- Working with CSV files
- Working with Excel (optional)
- Jinja2
- More APIs with Slack
- NAPALM
- Concurrency
- Unit Testing and CI/CD


Image Source: coffeeinmyveins.com

# Day 3 Review

- Modules, Packages, Imports, oh my!
  - Don’t forget to avoid namespace collisions — a file named “netmiko.py” is probably a bad idea!
- Pip and Virtual Environments
  - Keep your system(s) clean with venvs; pip freeze is your friend!
- TextFSM and Genie
  - Structured data > enormous strings
- Serialization
  - JSON and YAML are great for storing device information or data (plus they’re structured formats, and it turns out we like that…)
- APIs
  - Try to use good APIs whenever you can.

# Day 3 Review Exercises

Exercises:

{{ github_repo }}/day3/recap/exercise1.md

{{ github_repo }}/day3/recap/exercise2.md

# Subprocess - Integrating to the OS

```
import os

print()
print("Current working directory")
start_dir = os.getcwd()
print(os.getcwd())

print()
print("Path of module we are executing")
print(os.path.realpath(__file__))

print()
print("Is this a file?")
print(os.path.isfile(__file__))

print()
print("Change directory into /tmp")
os.chdir("/tmp")
print(os.getcwd())
```

# Subprocess - Integrating to the OS

```
import subprocess


def subprocess_wrapper(cmd_list):
    """Wrapper to execute subprocess including byte to UTF-8 conversion."""
    proc = subprocess.Popen(cmd_list, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
    std_out, std_err = proc.communicate()
    (std_out, std_err) = [x.decode("utf-8") for x in (std_out, std_err)]

    return (std_out, std_err, proc.returncode)


cmd_list = ["ls", "-a", "-l"]
std_out, std_err, return_code = subprocess_wrapper(cmd_list)
print()
print(f"Return Code: {return_code}")
print(std_out)
print()
```

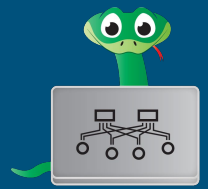
# Subprocess Exercises

Reference Examples:

{{ github_repo }}/day4/subprocess/collateral/

Exercises:

{{ github_repo }}/day4/subprocess/exercise1.md



PYTHON

FOR NETWORK ENGINEERS

# CSV Exercises

```
device_name,device_type,host,username,password
pynet-rtr1,cisco_ios,184.105.247.70,pyclass,my_pass
pynet-rtr2,cisco_ios,184.105.247.71,pyclass,my_pass
-----------------------------------
file_name = 'test_net_devices.csv'
with open(file_name) as f:
    read_csv = csv.DictReader(f)
    for entry in read_csv:
        print(entry)
```

# CSV Exercises

Reference Examples:

{{ github_repo }}/day4/csv_excel/collateral/

Exercises:

{{ github_repo }}/day4/csv_excel/exercise1.md

# Excel Exercises

```
from openpyxl import load_workbook


wb = load_workbook("excel_wb.xlsx")
print(f"Workbook Sheets: {wb.sheetnames}")
users_sheet = wb["Users"]
users_sheet.cell(row=5, column=3).value
```

# Excel Exercises

Reference Examples:

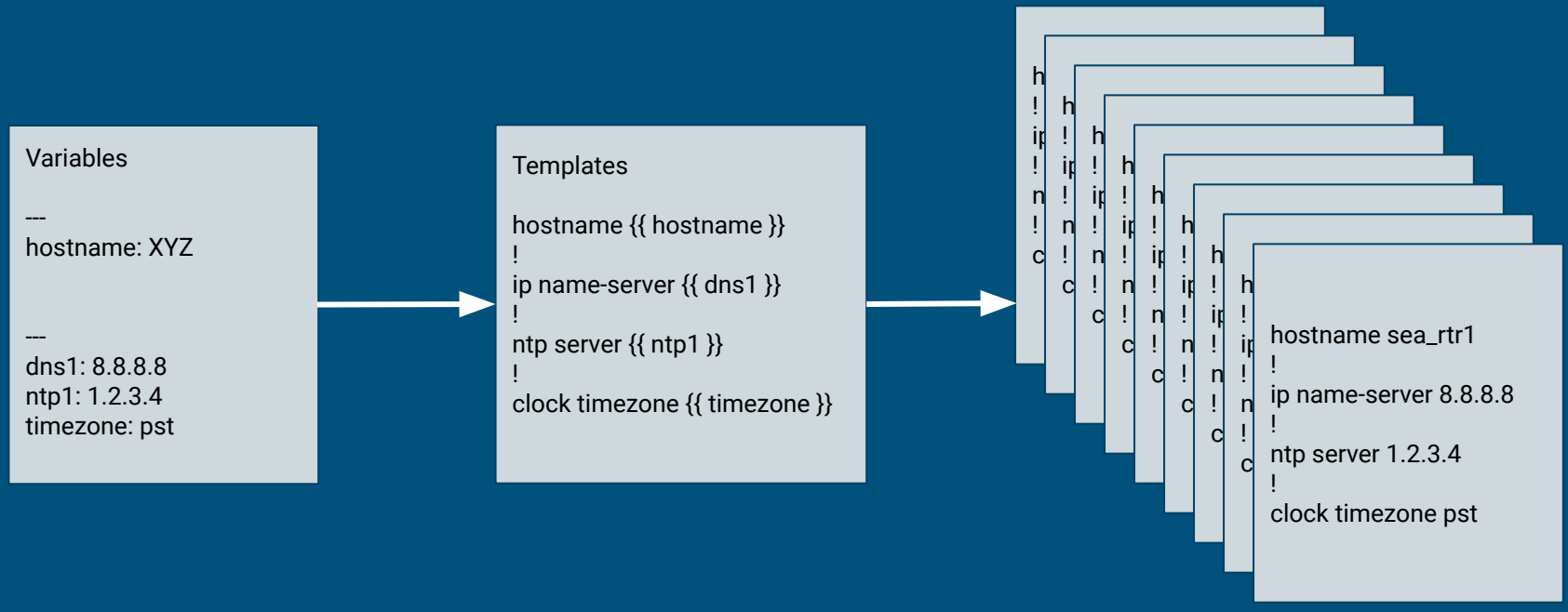
{{ github_repo }}/day4/csv_excel/collateral/

Exercises:

{{ github_repo }}/day4/csv_excel/exercise2.md

{{ github_repo }}/day4/csv_excel/exercise3.md

# Jinja2 (Templating)



# Jinja2 Templating


```
import jinja2

my_dict = {'a': 'whatever'}

my_template = '''
Some text
of something
{{ a }}
something
'''

t = jinja2.Template(my_template)
print(t.render(my_dict))
```

```
Some text
of something
whatever
something
```

# Jinja Exercises

Reference Examples:

{{ github_repo }}/day4/jinja/collateral/

Exercises:

{{ github_repo }}/day4/jinja/exercise1.md

# Jinja2 Templating
# Loading Template from a File


```
import jinja2

template_file = 'bgp_config.j2'
with open(template_file) as f:
    bgp_template = f.read()

my_vars = {
    'peer_as': '22',
    'neighbor1': '10.10.10.2',
    'neighbor2': '10.10.10.99',
    'neighbor3': '10.10.10.220',
}

template = jinja2.Template(bgp_template)
print(template.render(my_vars))
```

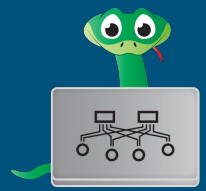
# Jinja Exercises

Reference Examples:

{{ github_repo }}/day4/jinja/collateral/

Exercises:

{{ github_repo }}/day4/jinja/exercise2.md



PYTHON

FOR NETWORK ENGINEERS

# Jinja2 Template - Environment

```
from __future__ import unicode_literals, print_function
from jinja2 import FileSystemLoader, StrictUndefined
from jinja2.environment import Environment

env = Environment(undefined=StrictUndefined)
env.loader = FileSystemLoader([".", "./templates/"])

my_vars = {"bgp_as": 22, "router_id": "1.1.1.1", "peer1": "10.20.30.1"}

template_file = "bgp_config.j2"
template = env.get_template(template_file)
output = template.render(**my_vars)
print(output)
```

# Jinja2 Conditionals


```
{% if SNMPv3 %}
access-list 98 remark *** SNMP ***
access-list 98 permit any
!
snmp-server view VIEWSTD iso included
snmp-server group READONLY v3 priv read VIEWSTD access 98
snmp-server user pysnmp READONLY v3 auth sha auth_key priv aes 128
encrypt_key
{% endif %}
```

# Jinja2 Loops


```
protocols {
    bgp {
        group external-peers {
            type external;
            {% for neighbor_ip, neighbor_as in my_list %}
                neighbor {{ neighbor_ip }} {
                    peer-as {{ neighbor_as }};
                }
            {% endfor %}
        }
    }
}
```

# Jinja2 Macros


- Reusable “function” in Jinja2

```
{% macro shutdown() %}
   Switchport access vlan 999
   Shutdown
{% end macro %}


{% for interface in interfaces %}
   {% if interface.state == “shutdown” %}
   {{ shutdown() }}
{% end for %}
```

# Jinja2 Includes / Hierarchy


- Include templates in other templates

```
{% include 'eos_ethernet_interface.j2' %}
{% include 'eos_loopback_interface.j2' %}
interface Management1
  ip address dhcp
!
ip routing
!
{% include 'eos_bgp.j2' ignore missing %}
!
{% include 'eos_ospf.j2'%}
```

- Ignore missing simply skips any “missing” templates

# Jinja2 Other


- Create Variables
  - `{% set my_variable = “something neat” %}`
- Whitespace
  - `{%- set my_variable = “something neat” %}` - Strip before
  - `{% set my_variable = “something neat” -%}` - Strip after
  - `{%+ set my_variable = “something neat” %}` - Preserve before
- Filters
  - Custom ways to modify variables
  - Example: `{{ my_string | center(80) }}`

# Jinja Exercises

Reference Examples:

{{ github_repo }}/day4/jinja/collateral/

Exercises:

{{ github_repo }}/day4/jinja/exercise3.md

# Slack API

- Tokens can be included in multiple locations:
  - Headers
  - Encoded in URL
  - In a payload
- POST - expects a “payload”
- Slack has lots of endpoints, things are not always strictly “RESTful”

```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()
```

# Slack API Exercises

Exercises:

{{ github_repo }}/day4/slack/exercise1.md

{{ github_repo }}/day4/slack/exercise2.md

# NAPALM

- Create a standard set of operations across a range of platforms.
  - Cisco IOS, NX-OS, IOS-XR
  - Arista EOS
  - Juniper Junos
- Configuration operations: Merge or Replace*
- Getters
  - config, arp, facts, interfaces, lldp_neighbors
  - bgp_neighbors, ntp_stats, get_route_to...
  - + More, not always available on all platforms, check the docs!

# NAPALM

| Method | Purpose |
| --- | --- |
| load_merge_candidate() | Populate a “candidate” configuration for a MERGE operation from string/file |
| load_replace_candidate() | Populate a “candidate” configuration for a REPLACE operation |
| compare_config() | Display diff between candidate and running configurations |
| discard_config() | Discard the candidate configuration |
| commit_config() | Commit the candidate config (using merge or replace depending on how config was loaded) |
| rollback() | If any changes were made, revert to initial state |
| get_*() | Get XYZ, arguments dependant upon getter |

# NAPALM Exercises

Reference Examples:

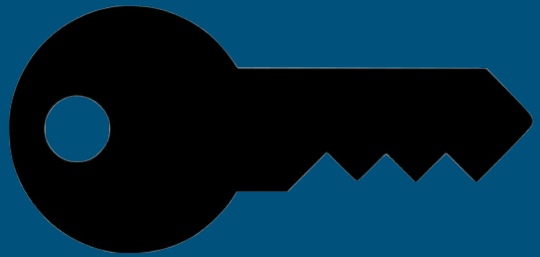
{{ github_repo }}/day4/napalm/collateral/

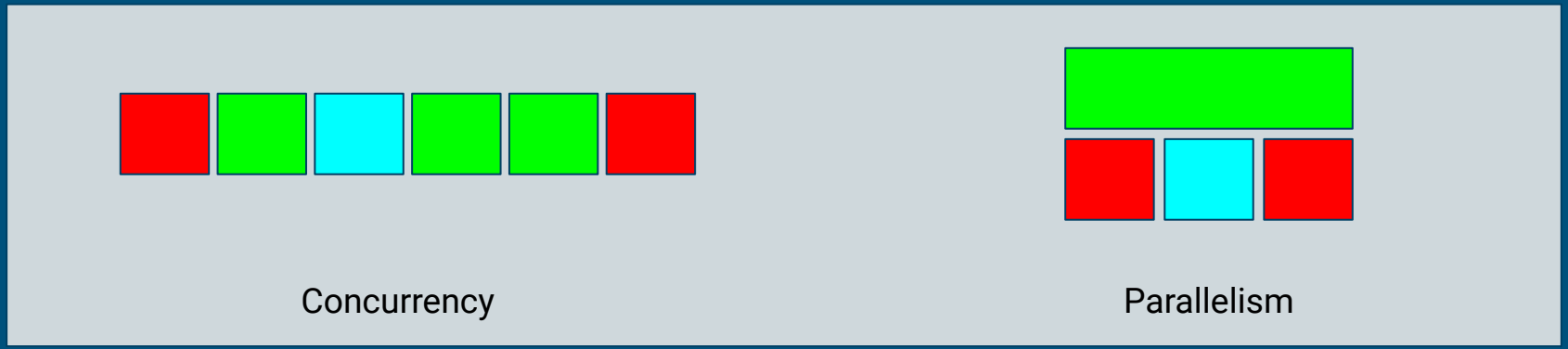
Exercises:

{{ github_repo }}/day4/napalmx/exercise1.md
{{ github_repo }}/day4/napalmx/exercise2.md
{{ github_repo }}/day4/napalmx/exercise3.md

# Concurrency

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





# 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

```
from concurrent.futures import ThreadPoolExecutor


pool = ThreadPoolExecutor(max_workers=8)
futures_threads = []
for _ in range(10):
    futures_threads.append(pool.submit(some_func))
```

# Concurrent Futures - ProcessPool

```
from concurrent.futures import ProcessPoolExecutor


pool = ProcessPoolExecutor(max_workers=8)
futures_procs = []
for _ in range(10):
    futures_procs.append(pool.submit(some_func))
```

# Concurrent Futures Exercises

Reference Examples:

{{ github_repo }}/day4/concurrency/collateral/

Exercises:

{{ github_repo }}/day4/concurrency/exercise1.md
{{ github_repo }}/day4/concurrency/exercise2.md

# Concurrent Futures - As Completed, Wait

```
from concurrent.futures import ProcessPoolExecutor, as_completed, wait


pool = ProcessPoolExecutor(max_workers=8)
futures_procs = []
for _ in range(10):
    futures_procs.append(pool.submit(some_func))
for proc in as_completed(futures_procs):
    print(proc.result())
# wait(futures_procs)
```

# Concurrent Futures Exercises

Reference Examples:

{{ github_repo }}/day4/concurrency/collateral/

Exercises:

{{ github_repo }}/day4/concurrency/exercise3.md

PYTHON

FOR NETWORK ENGINEERS

# Unit Testing

- What is “unit testing”
  - Testing components of your code to ensure they do what you think they should do.
  - Generally test modular parts of your code (functions, methods, objects).
- Why care about it?
  - Bugs *will* happen
  - Forces you to think about Single Responsibility Principle and DRY.
  - Increase confidence that unintended consequences didn’t happen when making changes.
  - Speed up development process by reducing manual testing and rework.

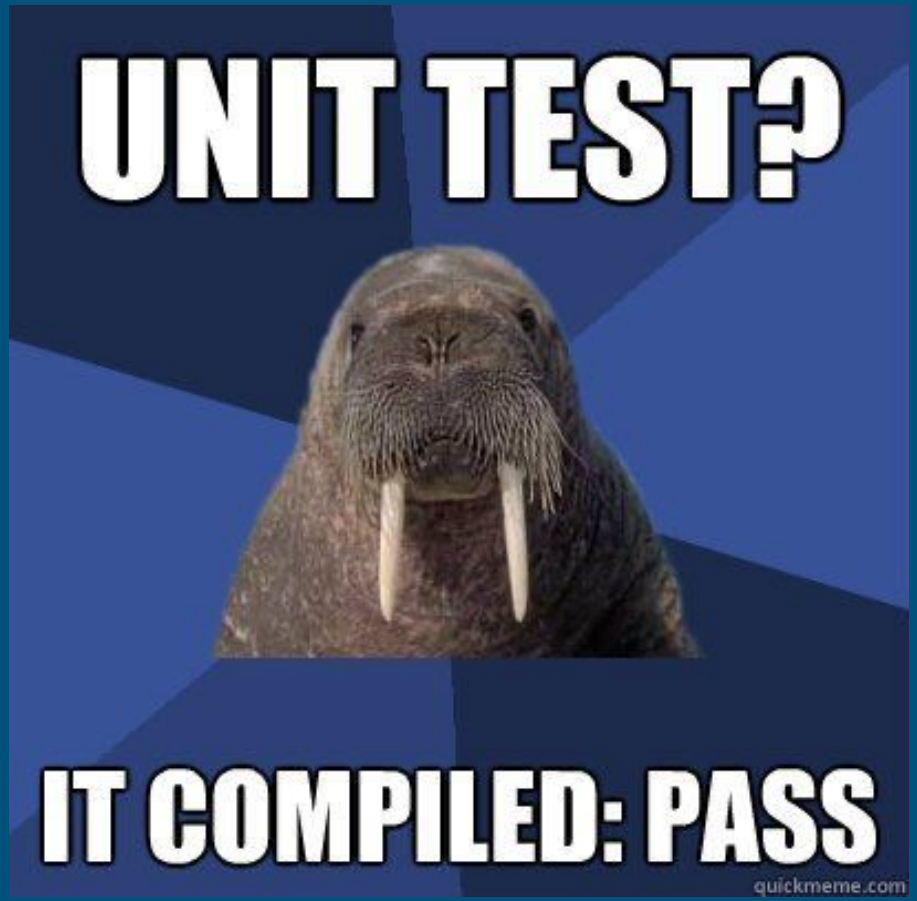

*But my unit tests work!*

# Pytest

```
import pytest

# Functions
def func(x):
    return x + 1

# Tests
def test_answer():
    assert func(3) == 4
```



# Pytest


```
$ python -m pytest -s -v ./test_simple.py

==================== test session starts ====================================
platform linux -- Python 3.6.8, pytest-4.6.2, py-1.8.0, pluggy-0.12.0 --
/home/user/VENV/py3_venv/bin/python36
cachedir: .pytest_cache
rootdir: /home/user/my_repo/testing_example/pytest_dir
plugins: pylama-7.7.1, f5-sdk-3.0.21
collected 2 items

test_simple.py::test_answer PASSED
test_simple.py::test_answer2 PASSED

================= 2 passed in 0.01 seconds ==================================
```

# Pytest Fixtures


```
@pytest.fixture(scope="module")
def netmiko_connect():
    cisco1 = {
        'device_type': 'cisco_ios',
        'ip':   '184.105.247.70',
        'username': 'pyclass',
        'password': getpass()
    }
    return ConnectHandler(**cisco1)
```

# Pytest Fixtures


```
def test_prompt(netmiko_connect):
    print(netmiko_connect.find_prompt())
    assert netmiko_connect.find_prompt() == 'pynet-rtr1#'

def test_show_version(netmiko_connect):
    output = netmiko_connect.send_command("show version")
    assert 'Configuration register is 0x2102' in output
```

# Unit Testing Exercises


Reference Examples:

{{ github_repo }}/day4/testing/collateral/

Exercises:

{{ github_repo }}/day4/testing/exercise1.md
{{ github_repo }}/day4/testing/exercise2.md

# Continuous Integration

*If it doesn't happen automatically; you can't rely on it.*

Travis CI

GitLab

Azure Pipelines

circleci

# Continuous Integration

Travis CI

- Testing is cool, but *automated* testing is cooler!
- Typically CI creates a test environment
  - Run on as many supported OSes as possible/needed
  - Install any requirements
  - Compile
  - Lint
  - Run unit tests

```
---
dist: xenial
language: python
python:
  - "3.6"
  - "3.7"
install:
  - pip install -r requirements.txt
script:
  - pylama .
  - black --check .
  - ./check_line_lengths.sh
  - py.test -s -v day4/test_ex1.py
```

# Unit Testing Exercises

Exercises:

{{ github_repo }}/day4/ci/exercise1.md

# The end...

ktbyers@twb-tech.com
Twitter: @kirkbyers

carl@twb-tech.com
Twitter: @carlrmontanari