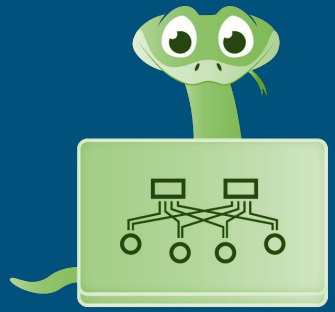


PYTHON

FOR NETWORK ENGINEERS

Onsite Training Session
July 2020

$ whoami

Kirk Byers
Network Engineer:
CCIE #6243 (emeritus)

Programmer:
Netmiko
NAPALM
Nornir

Teach Python, Ansible, Nornir in a Network Automation context

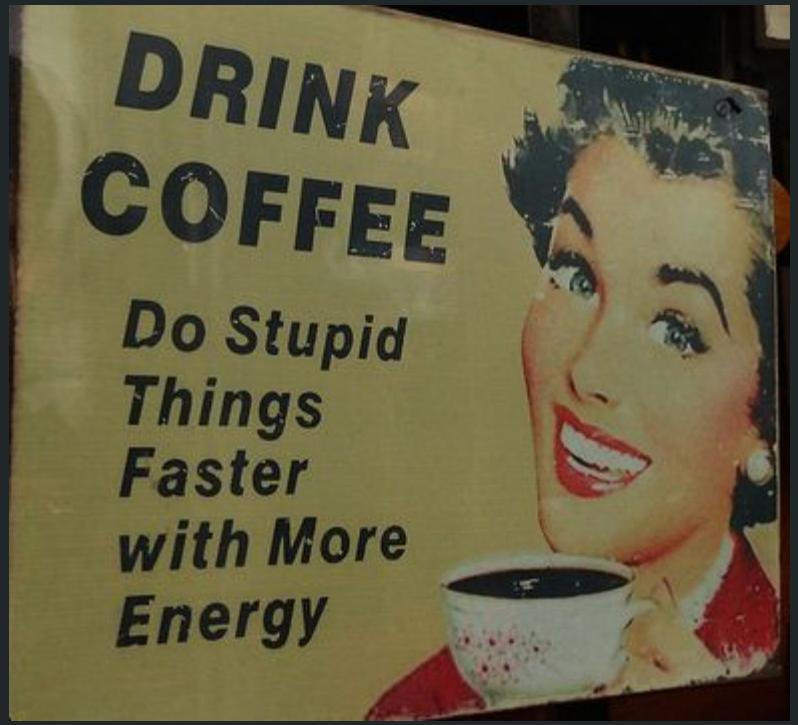
General:

July 21, Day1 (Tue)
July 22, Day2 (Wed)
July 23, Day3 (Thu)

July 28, Day4 (Tue)
July 29, Day5 (Wed)

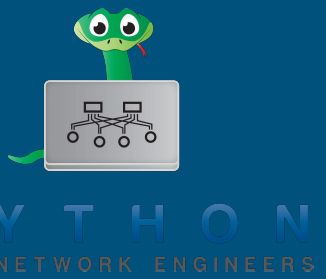
Focused/Minimize Distractions

The exercises are important.



Flickr: Ben Sutherland

Day1 Schedule



Course introduction
Why Python?

Working with Git

Python Fundamentals Review

- Flow-control
- Data Structures
- Functions

Linting

pytest and testing

Git

- 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
- Creating a repository on GitHub
- Cloning a repository
- git init
- Files have four different states: untracked, modified, staged, committed

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 diff *# what changed*

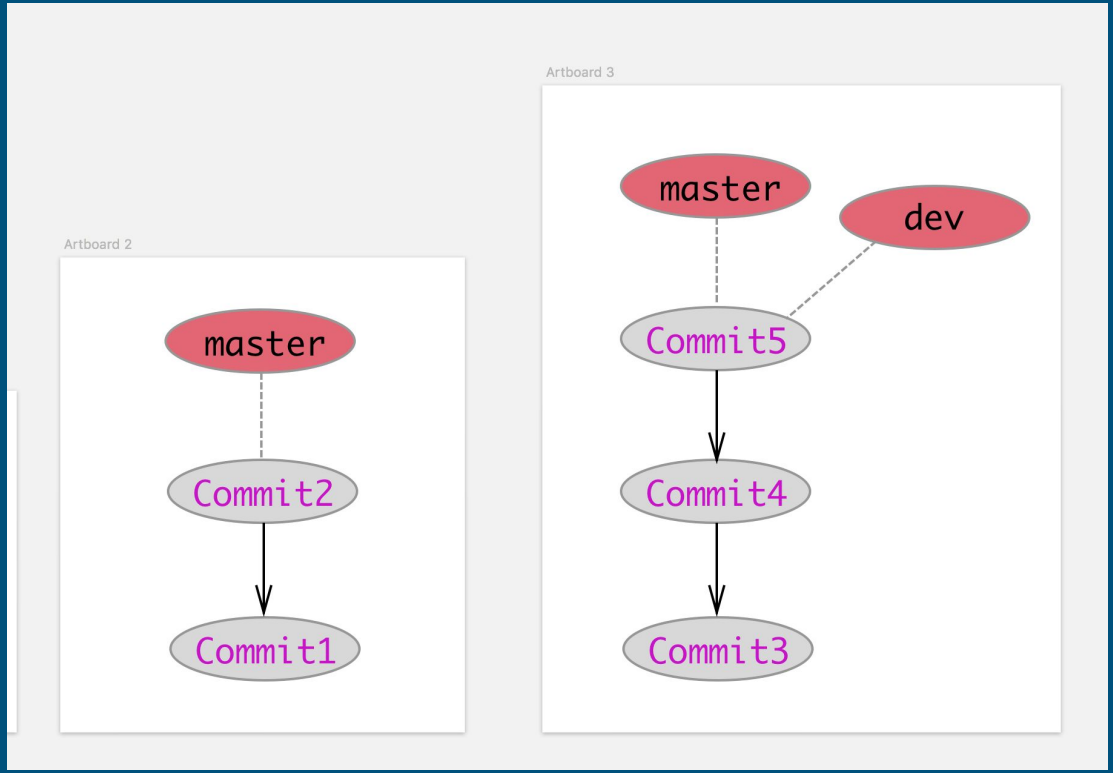
Git Push & Pull

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

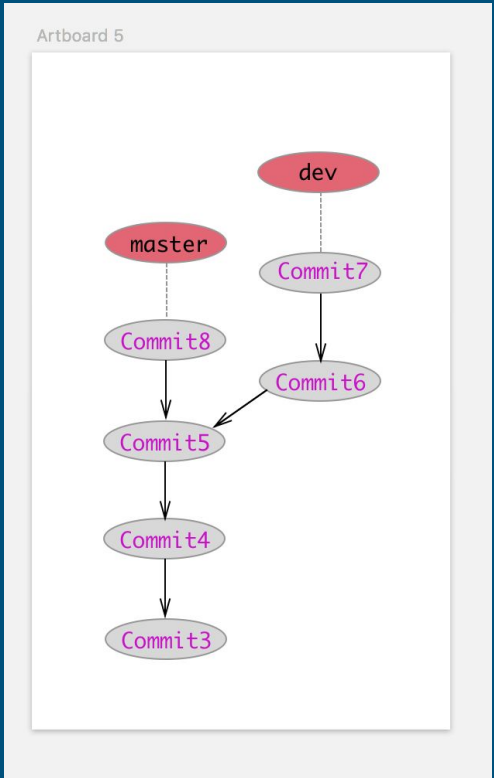
- git pull / git push
  - git remote -v
  - git remote add
  - git branch -vv

Reference Commands:
{{ github_repo }}/git_notes/git_commands.MD

Git Branches



Git Branches



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.

Git Exercises

Reference Commands:

{{ github_repo }}/git_notes/git_commands.MD

Exercises:
./day1/git/git_ex1.txt
./day1/git/git_ex2.txt

VI in five minutes

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)

VI in five minutes

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

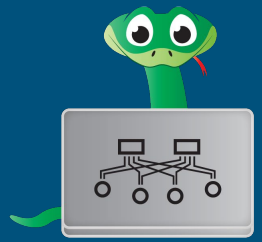
:wq - saves file and exits
:q! - exits WITHOUT saving

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

REMEMBER:
<esc> is your friend

Why Python?

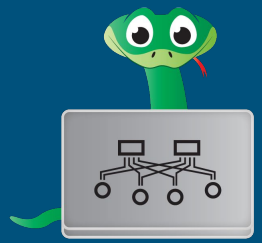
- Widely supported (meaning lots of library support)
- Easily available on systems
- Language accommodates beginners through advanced
- Maintainable
- Allows for easy code reuse
- High-level



PYTHON
FOR NETWORK ENGINEERS

Python Review

- Data Structures
- Flow Control
- Functions



PYTHON
FOR NETWORK ENGINEERS

Python Characteristics

Indentation matters.

Use spaces not tabs.

Python programmers are particular.

Py2 or Py3. *# The battle is now over: use Python3.*

Python2 support ended on Jan1, 2020.

General Items

The Python interpreter shell
Assignment and variable names
Python naming conventions
Printing to standard out/reading from standard in
Creating/executing a script
Quotes, double quotes, triple quotes
Comments
dir() and help()

Lists

Zero-based indices

.append()

.pop()

.join()

List slices

Tuple

Copying a list

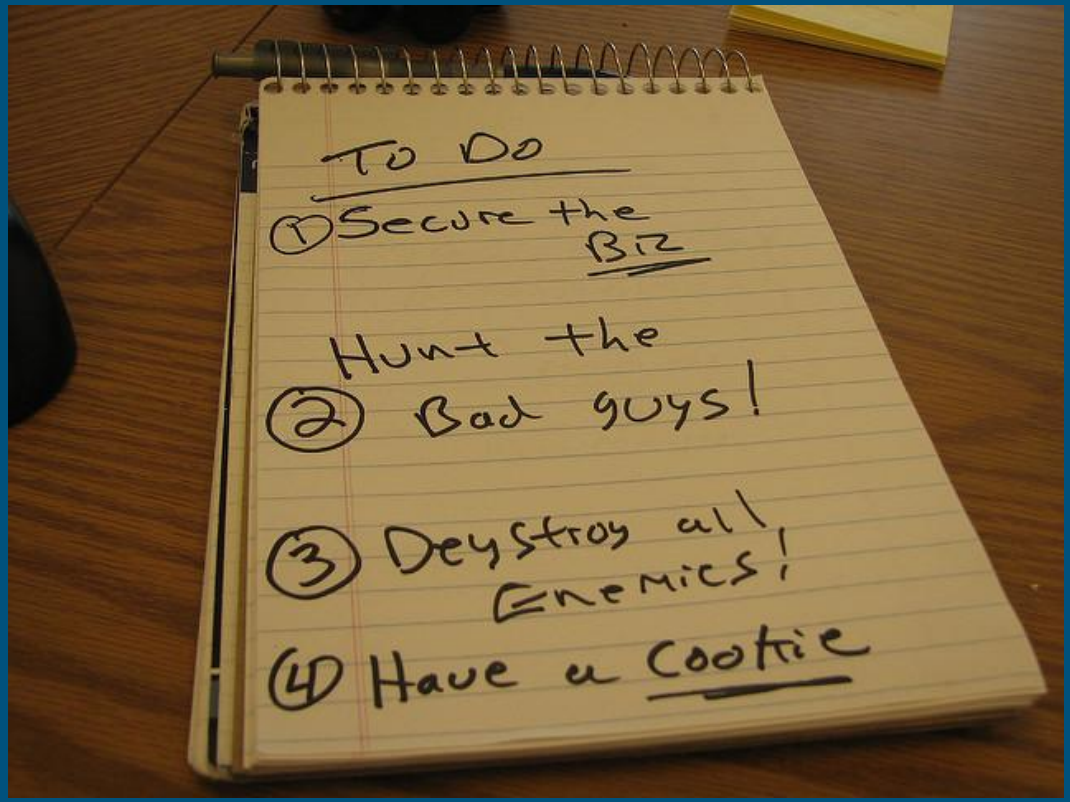


Photo: Purple Slog (Flickr)

Exercises:
./day1/py_lists/lists_ex1.txt
./day1/py_lists/lists_ex2.txt

Dictionaries

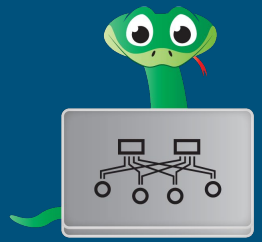
- Creating
- Updating
- get()
- pop()
- Iterating over keys
- Iterating over keys and values

Exercises:
./day1/py_dict/dict_ex1.txt

Photo: Holger Zscheyge (Flickr)

Conditionals

```
if a == 15:
    print("Hello")
elif a >= 7:
    print("Something")
else:
    print("Nada")
```



PYTHON
FOR NETWORK ENGINEERS

Loops

- for
- while
- break
- continue
- range(len())
- enumerate

Photo: Mário Monte Filho (Flickr)

For/while syntax

```
for my_var in iterable:
    print(my_var)
```

```
i = 0
while i < 10:
    print(i)
    i += 1
```

Exercises:
./day1/py_loops/loops_ex1.txt
./day1/py_loops/loops_ex2.txt

Exception Handling

```
try:
    my_dict['missing_key']
except KeyError:
    do_something
```

- Trying to gracefully handle errors.
- finally: - always ran if you have a cleanup condition.

Exercises:
./day1/py_except/except_dict_ex1.txt

Exercise:

Exercises:
./day1/exercise_show_ver/for_cond_show_ver_ex1.txt

Show Version Exercise

---

a. Read a show version output from a router (in a file named, “show_version.txt”.

b. Find the router serial number in the output.

c. Parse the serial number and return it as a variable. Use .split() and substr in str to accomplish this.

Functions:

- Defining a function
- Positional arguments
- Named arguments
- Mixing positional and named arguments
- Default values
- Passing in *args, **kwargs
- Functions and promoting the reuse of code

Exercises:
./day1/py_func/func_ex1.txt
./day1/py_func/func_ex2.txt
./day1/py_func/func_ex3.txt
./day1/py_func/func_ex4.txt

Python Regular Expressions

import re

Other re methods

re.split()
re.sub()
re.findall()

Exercises:
./day1/py_regex/regex_ex1.txt
./day1/py_regex/regex_ex2.txt

re.search(pattern, string)

- always use raw strings
- re.M/re.MULTILINE
- re.DOTALL
- re.I
- Parenthesis to retain patterns
- greedy/not greedy (.*?)

match.group(0)
match.groups()
match.groupdict()

Named patterns

(?P<software_ver>Ver.*)

Regular Expression Resources

Regular Expression Tutorial
https://regexone.com/lesson/introduction_abcs
This is a good resource if you are new to regular expressions.

Online Regular Expression Tester
https://regex101.com/
Select 'Python' on the left-hand side.

Python Regular Expression HowTo
https://docs.python.org/2/howto/regex.html
This is a good overview of regular expression special characters.
Start at the very top of the page and read through the 'Repeating Things' section.

Python Code Structure:

- Imports at top of the file
- CONSTANTS
- Functions / classes
- if __name__ == “__main__”:
- Main code or main() function call

Exercises:
./day1/py_reuse/reuse_ex1.txt

Modules and Packages

Python Module

A Python file that you can import into another Python program

Example, storing device definition in an external file.

Python Package

An importable Python directory

__init__.py

Exercises:
./day1/py_reuse/reuse_ex2.txt
./day1/py_reuse/reuse_ex3.txt

Review Exercise

Process the 'show_ip_int_brief.txt' file and create a data structure from it.

1. Create a dictionary of dictionaries.
2. The keys for the outermost dictionary should be the interface names.
3. The value corresponding to this interface name is another dictionary with the fields
   'ip_address', 'line_status', and 'line_protocol'.
4. Use pretty-print to print out your data structure.

Your output should be similar to the following:

```
{'FastEthernet0': {'ip_address': 'unassigned',
                   'line_protocol': 'down',
                   'line_status': 'down'},
 ... }
```

Exercises:
./day1/exercise_review/review_ex1.txt

Review Exercise

Process the 'show_arp.txt' file and create a data structure from it.

1. Create a dictionary where the keys are the ip addresses and the corresponding values are the mac-addresses.

2. Create a second dictionary where the keys are the mac-addresses and the corresponding values are the ip addresses.

3. Use pretty print to print these two data structures to the screen.

Exercises:
./day1/exercise_review/review_ex2.txt

Python Linters

Auto formatting with Python Black

pylint or pycodestyle

Consistency and conventions make your life easier.

Finds obvious errors. Finds problems you might not be aware of (reuse of builtins).

pylint my_file.py
pycodestyle my_file.py
pylama my_file.py

Unit testing with pytest

```python
# Functions
def increase_by_one(x):
    return x + 1

# Tests
def test_simple_case():
    assert increase_by_one(3) == 4

def test_various_cases():
    assert increase_by_one(-1) == 0
    assert increase_by_one(0) == 1
    assert increase_by_one(10) == 11
    assert increase_by_one(-10) == -9
```

1. Test the components of your code (functions, methods, classes).
2. Raises confidence that your code is working properly.
3. Allows you to change the code with a higher degree of confidence that you didn't introduce new errors.

pytest - How to get started?

```python
# Functions
def increase_by_one(x):
    return x + 1


# Tests
def test_simple_case():
    assert increase_by_one(3) == 4


def test_various_cases():
    assert increase_by_one(-1) == 0
    assert increase_by_one(0) == 1
    assert increase_by_one(10) == 11
    assert increase_by_one(-10) == -9
```

1. Create a file named test_something.py.
2. In this file define functions named test_thing_you_are_testing.
3. Add assert statements inside the tests (if the assert evaluates to True, your test will pass; otherwise, it will fail).

pytest - Run the test.


```
(py3_venv) [student20@onslab1a pytest_dir]$ py.test -s -v test_simple.py
==================================== test session starts ====================================
platform linux -- Python 3.6.10, pytest-5.4.3, py-1.9.0, pluggy-0.13.1 -- /home/student20/VENV
/py3_venv/bin/python3
cachedir: .pytest_cache
rootdir: /home/student20/pynet-ons/unittest_example/pytest_dir
plugins: pylama-7.7.1, f5-sdk-3.0.21
collected 2 items

test_simple.py::test_simple_case PASSED
test_simple.py::test_various_cases PASSED

==================================== 2 passed in 0.01s =====================================
```

pytest - Run the test.

1. Use the -v (verbose) argument and the -s (std-in) arguments.
2. Change directory into the directory with your test files.
3. You can specify: `py.test -s -v .`
4. This will search for all files named test_ and inside those files for any functions named test_.
5. There is also a way you can test clases.
6. If you specify a directory py.test will recurse down into subdirectories.

pytest - Skip tests


```python
@pytest.mark.skip(reason="Unable to test")
def test_example():
    assert True

@pytest.mark.skipif(sys.version_info < (3, 7), reason="requires python3.7 or later")
def test_skip_os_ver():
    print(sys.version_info)
    assert True
```

pytest - parametrize


```python
# Functions
def increase_by_one(x):
    return x + 1

def test_various_cases():
    assert increase_by_one(-1) == 0
    assert increase_by_one(0) == 1
    assert increase_by_one(10) == 11
    assert increase_by_one(-10) == -9

@pytest.mark.parametrize("number, result", [(-1, 0), (0, 1), (10, 11), (-10, -9)])
def test_various_cases_params(number, result):
    assert increase_by_one(number) == result
```

pytest - *These examples show you the mechanics, but how to approach the problem?*

1. Have one clear purpose to your functions and methods.
2. Test verifies that one clear purpose.
3. Layer unit tests, integration tests, and systems tests.
4. Don’t forget linting tools.
5. Use code-coverage tools to look for how well you are testing.
6. Use CI-CD tools to integrate testing into your workflow.

But my unit tests work...

1. Don’t miss the forest for the trees.
2. How to test things that interface to remote systems (mock, test systems).
3. How to apply this all in a network engineering context.

Exercises:
./day1/testing/unittest_ex1.txt
./day1/testing/unittest_ex2.txt

Reference Material in:
{{ github_repo }}/unittest_example