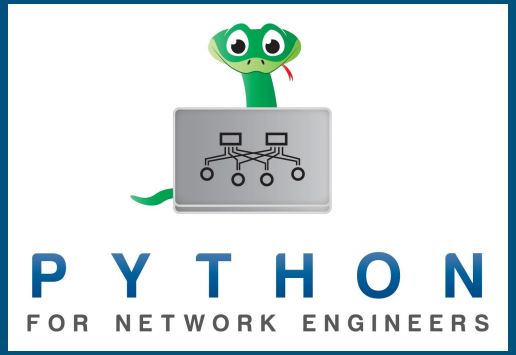
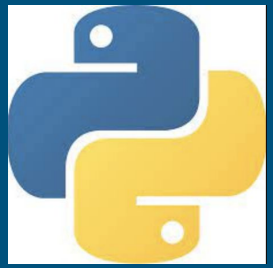




Live Training Session
December 2021

$ whoami

Kirk Byers
Network Engineer
CCIE #6243 (emeritus)

Programmer
Netmiko
NAPALM
Nornir

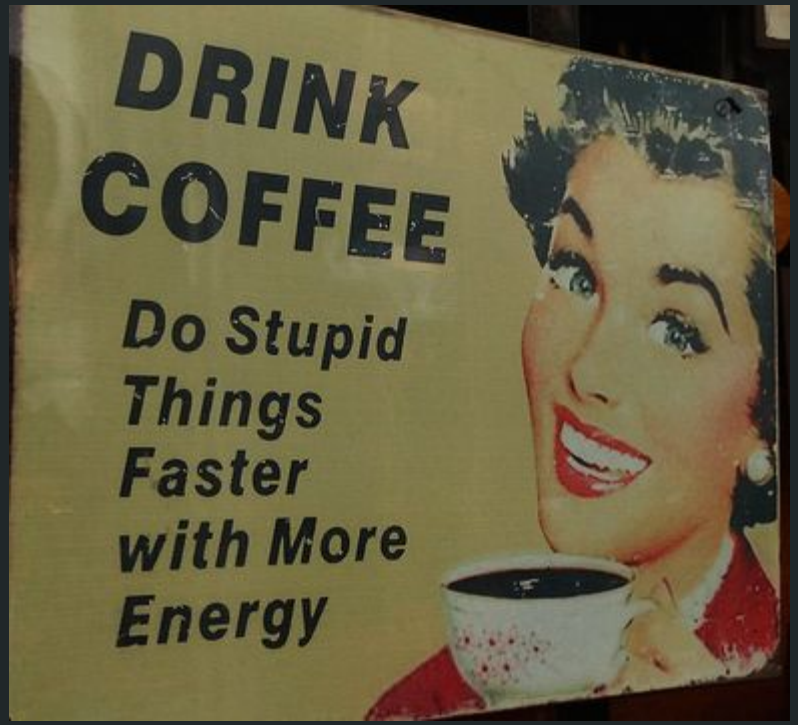
Teach Python, Ansible, Nornir in a Network Automation context

General:

Dec 7, Day1 (Tue) / 8AM - 4PM Central
Dec 8, Day2 (Wed)
Dec 9, Day3 (Thu)
Dec 10, Day4 (Fri)

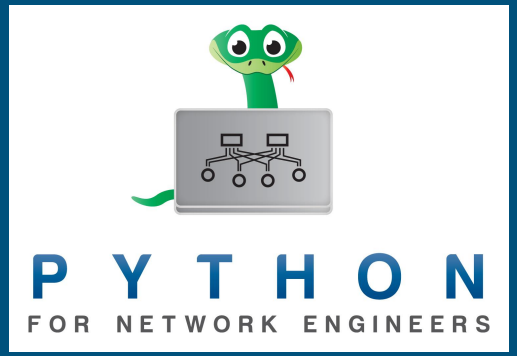
Focused/Minimize Distractions

The exercises are important.



Flickr: Ben Sutherland

Day1 Schedule



Course introduction
Why Python?

Working with Git

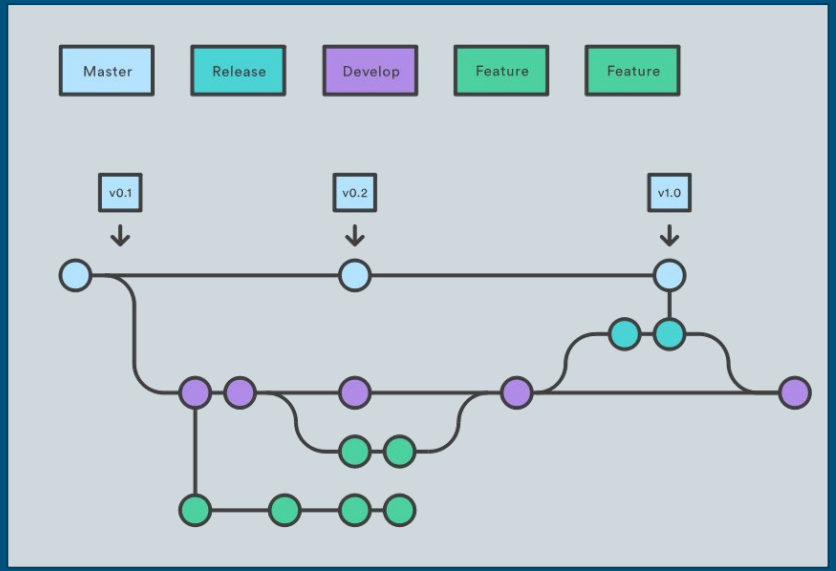
Python Fundamentals Review

- Data Types/Data Structures
- Flow-control
- Exceptions
- Functions

Netmiko (Part1)

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*

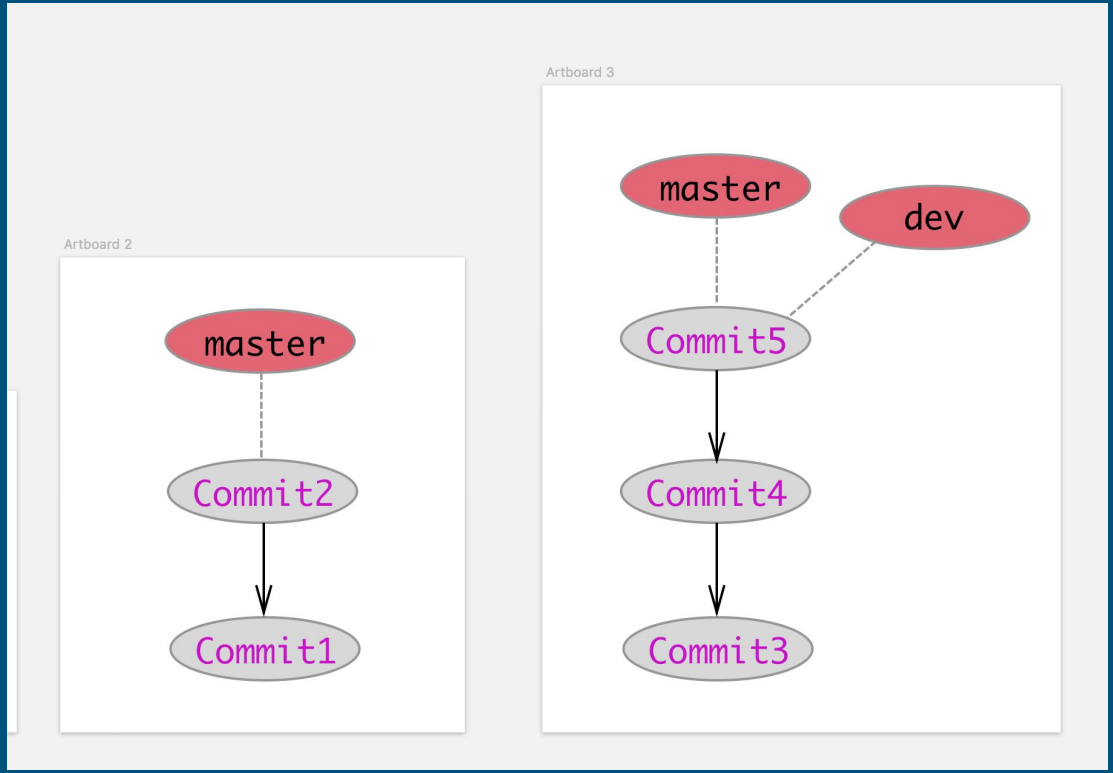
Git Push & Pull

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

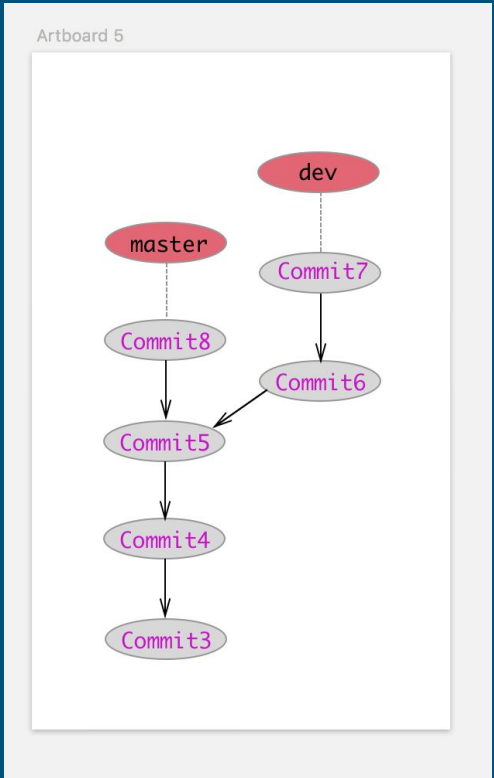
- git pull / git push
- git remote -v
- git remote add
- git branch -vv

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

Git Branches



Git Branches



Git Branches

Creating a branch

- git checkout -b dev origin/main
- 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.

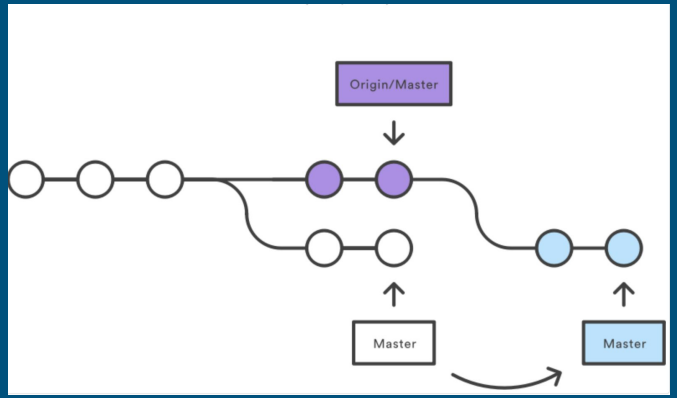
$ cat test2.py

```
------------
    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.

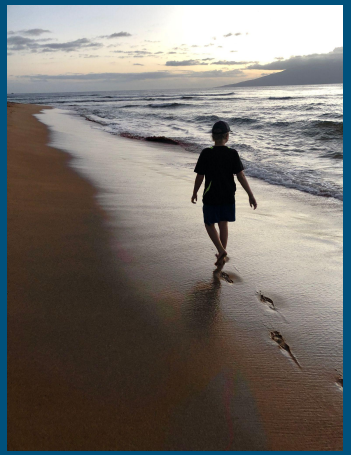


Git Exercises

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

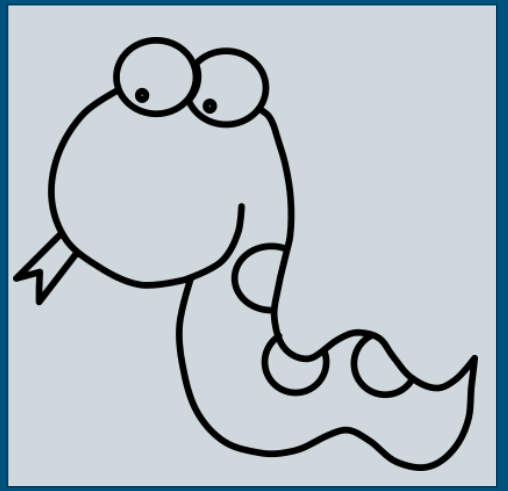
:wq - saves file and exits
:q! - exits WITHOUT saving

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

REMEMBER:
<esc> is your friend

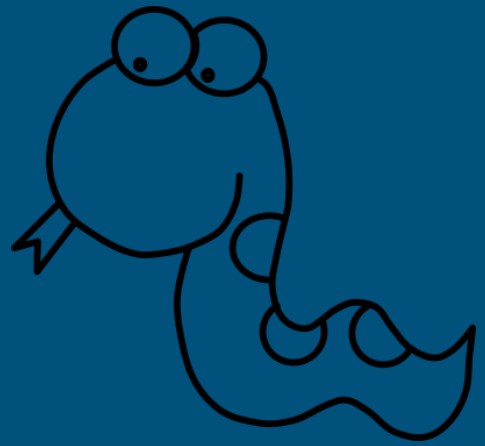
Why Python?

- 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 Review

- Strings / Numbers
- Lists
- Conditionals
- Loops
- Dictionaries
- Files
- Exceptions
- Functions
- Regular Expressions (optional)



Python Characteristics

Indentation matters.

Use spaces not tabs.

Python programmers are particular.

Py3 *# The battle is 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()

Strings

- String methods
- Chaining
- split()
- strip()
- substr in string
- unicode
- raw strings
- format() method
- f-strings

```
In [6]: print(emoji.emojize(':grinning_face:'))
😀

In [7]: print(emoji.emojize(':grinning_squinting_face:'))
😆

In [8]: print(emoji.emojize(':smiling_face:'))
☺️

In [9]: print(emoji.emojize(':guide_dog:'))
🦮

In [10]: print(emoji.emojize(':thumbs_up:'))
👍
```

Exercises:
./day1/py_strings/str_ex1.txt
./day1/py_strings/str_ex2.txt

Numbers

Integers
Floats
Math Operators (+, -, *, /, **, %)

Exercises:
./day1/py_numbers/numbers_ex1.txt

Lists

```
In [1]: my_list = ["foo", 1, "hello", [], None, 2.7]
```

Zero-based indices

.append()

.pop()

.join()

List slices

Tuple

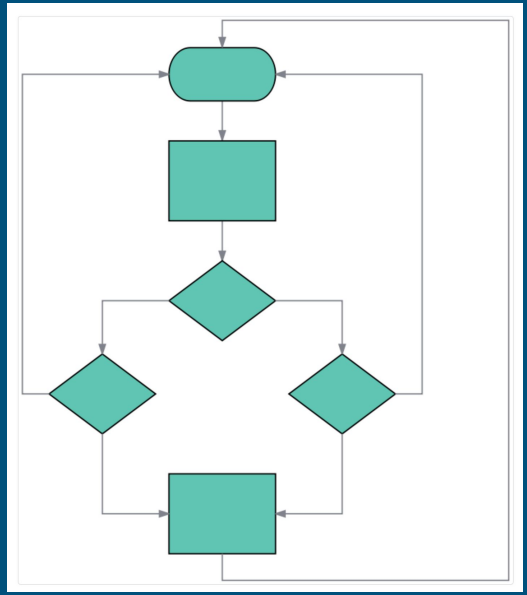
Copying a list

```
In [3]: my_list[0]
Out[3]: 'foo'

In [4]: my_list[-1]
Out[4]: 2.7
```

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

Conditionals

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



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

Dictionaries

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

```
In [1]: my_devices = {
   ...:     "sf-rtr1": {
   ...:         "hostname": "cisco1.lasthop.io",
   ...:         "device_type": "cisco_ios",
   ...:         "username": "pyclass",
   ...:         "password": "invalid",
   ...:     }
   ...: }

In [2]: type(my_devices)
Out[2]: dict
```

Exercises:
./day1/py_dict/dict_ex1.txt

Booleans and None

Boolean operators (and, or, not)

is

Truish

Comparison operators (==, !=, <, >, >=, <=)

None

```
my_value = None

val1 = True
val2 = False

if val1 and val2:
    print("Hello")

if val1 or val2:
    print("World")

if my_value is None:
    print("Whatever")
```

Writing to a file/reading from a file:

```
with open(file_name, "w") as f:
    f.write(output)
```

```
with open(file_name) as f:
    output = f.read()
```

Exercises:
./day1/py_files/files_ex1.txt

Exception Handling

- Trying to gracefully handle errors.
- finally - always runs

Exercises:
./day1/py_except/except_dict_ex1.txt

```
my_dict = {}

try:
    my_dict["foo"]
except KeyError:
    print("Exception happened...handled it gracefully")
finally:
    print("Always runs...regardless of exception")
```

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:

```python
def my_func(arg1, arg2, arg3=None):
    print("This is a function")
    print(f"arg1 value --> {arg1}")

    return arg1 + arg2

# Call the function
my_func(22, 33)
```

- 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/3.9/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.

Netmiko

Netmiko is a multi-vendor networking library based on Paramiko.

https://github.com/ktbyers/netmiko

PYTHON

FOR NETWORK ENGINEERS

Netmiko Vendors

- Currently there are (very) roughly 84 different platforms supported by Netmiko.
- Three different categories of supported platform (regularly tested, limited testing, experimental).

https://ktbyers.github.io/netmiko/PLATFORMS.html

Regularly tested
Arista vEOS
Cisco ASA
Cisco IOS
Cisco IOS-XE
Cisco IOS-XR

Regularly tested
Cisco NX-OS
Cisco SG300
HP ProCurve
Juniper Junos
Linux

Key Netmiko Methods

.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()	… 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)
FileTransfer Class	SCP files to/from devices

Netmiko example

```
#!/usr/bin/env python
from getpass import getpass
from netmiko import ConnectHandler

password = getpass()

device = {
    "device_type": "arista_eos",
    "host": "arista1.lasthop.io",
    "username": "pyclass",
    "password": password,
}

net_connect = ConnectHandler(**device)
print(net_connect.find_prompt())
net_connect.disconnect()
```

Reference Material in:

{{ github_repo }}/netmiko_example

Netmiko ‘show’ command


```
net_connect = ConnectHandler(**device)
output = net_connect.send_command("show lldp neighbors")
net_connect.disconnect()

print("-" * 50)
print(output)
print("-" * 50)
```

Netmiko multiple devices

```
arista4 = {
    "device_type": "arista_eos",
    "host": "arista4.lasthop.io",
    "username": "pyclass",
    "password": password,
}

for device in (arista1, arista2, arista3, arista4):
    net_connect = ConnectHandler(**device)
    output = net_connect.send_command("show ip int brief")

    print()
    print(f"Host: {net_connect.host}:{net_connect.port}")
    print("-" * 50)
    print(output)
    print("-" * 50)
    net_connect.disconnect()
```

Exercises:
./day1/netmiko/netmiko_ex1.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 rich.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