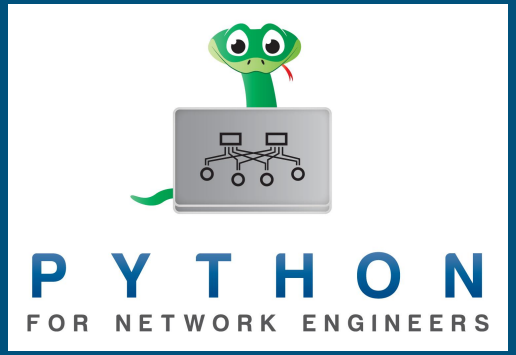
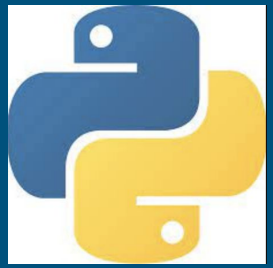




Virtual Training Session
October 2022

$ whoami

Kirk Byers
Network Engineer
CCIE #6243 (emeritus)

Programmer
Netmiko
NAPALM
Nornir

Teach Python, Ansible, Nornir in a Network Automation context

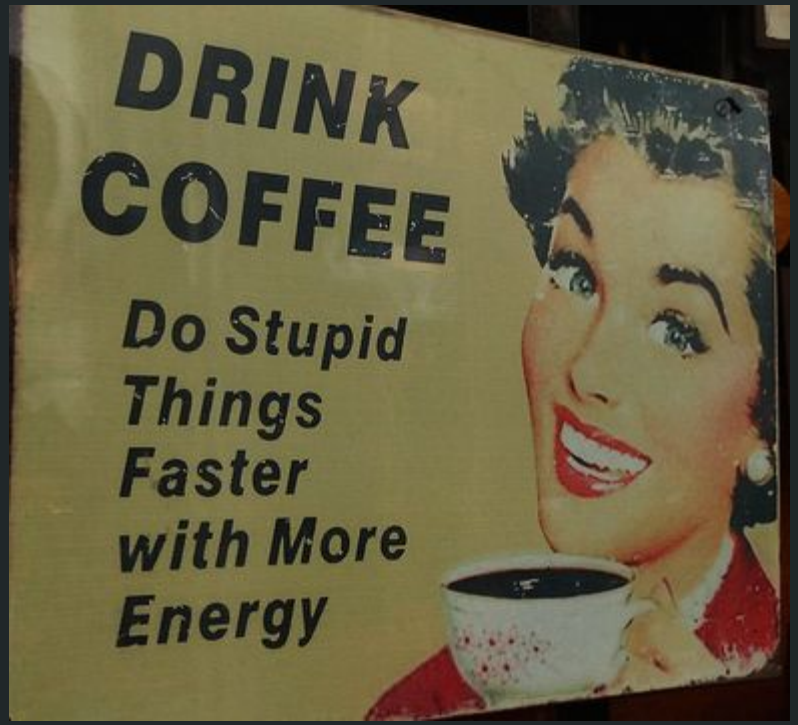
General:

Virtual-Training Day
Oct 5th, Day1 (Wed) / 9AM - 4:30PM Central

Two-Day Live Training
Nov 15th - 16th (Tues/Weds)

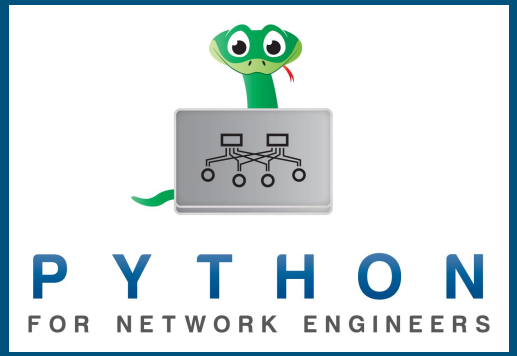
Focused/Minimize Distractions

The exercises are important.



Flickr: Ben Sutherland

Virtual-Day Training Schedule



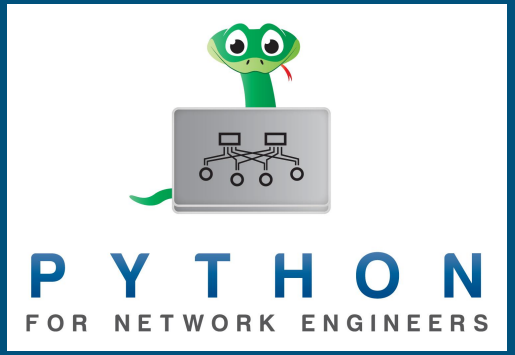
Course Introduction

Environment Setup
- Python Install
- Git Install
- Create GitHub Account
- VSCode Install

VS Code Setup

Git Review

Virtual-Day Training Schedule



Why Python?

Python Fundamentals
- Strings / Numbers
- Lists
- Conditionals
- Loops
- Dictionaries
- Files
- Exceptions
- Functions

Python Fundamentals
- sys.path and $PYTHONPATH
- Libraries
- PIP
- Linters and VS Code
- Intro to Classes and Objects (optional)
- More Data Structures (optional)

Environment Setup

Things that should be working at this point:

```
$ python3
Python 3.10.7 (v3.10.7:6cc6b13308, Sep  5 2022, 14:02:52) [Clang 13.0.0 (clang-1300.0.29.30)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

```
$ git version
git version 2.37.0 (Apple Git-136)
```

```
 $ git config --global user.email
ktbyers@twb-tech.com
```

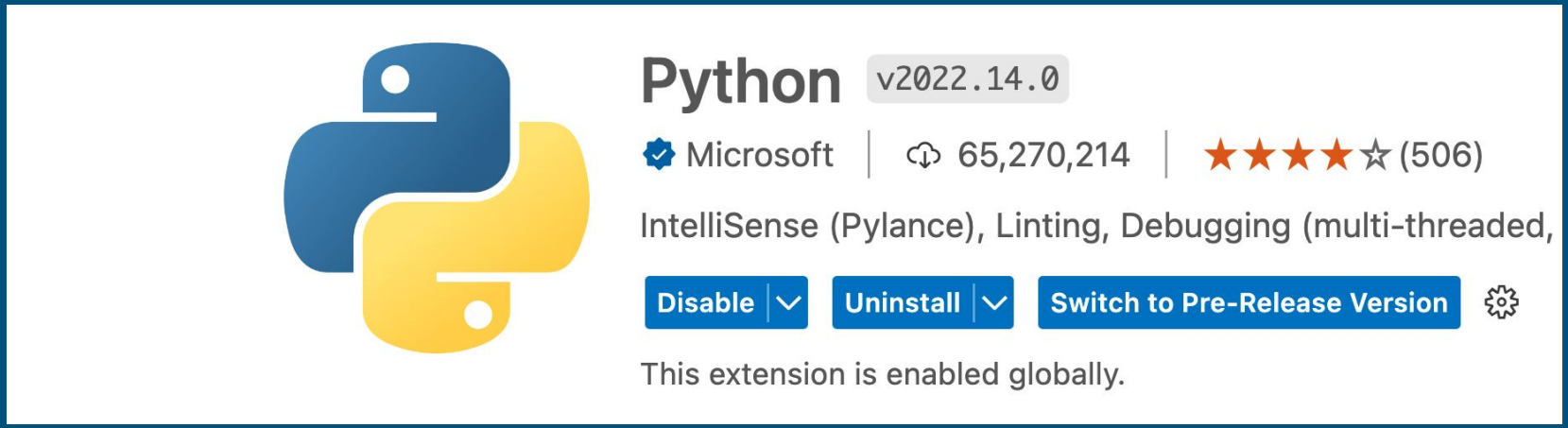
Environment Setup

Things that should be working at this point:

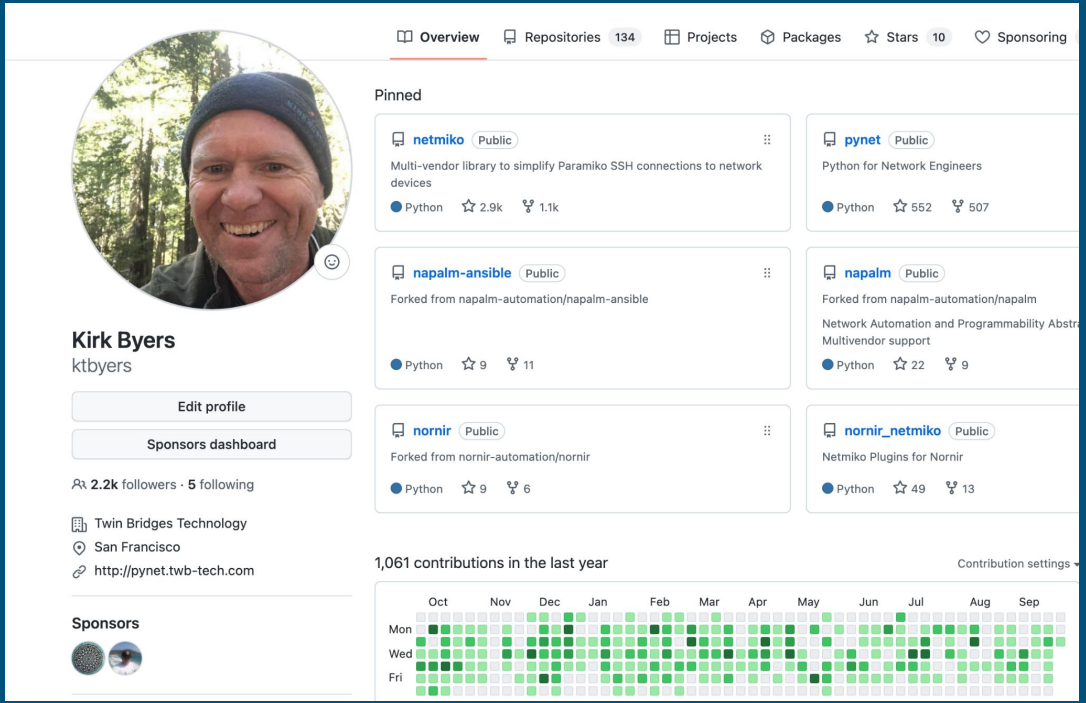
$ git config --global user.name
Kirk Byers

VS Code Setup

Install Python extension for VS Code

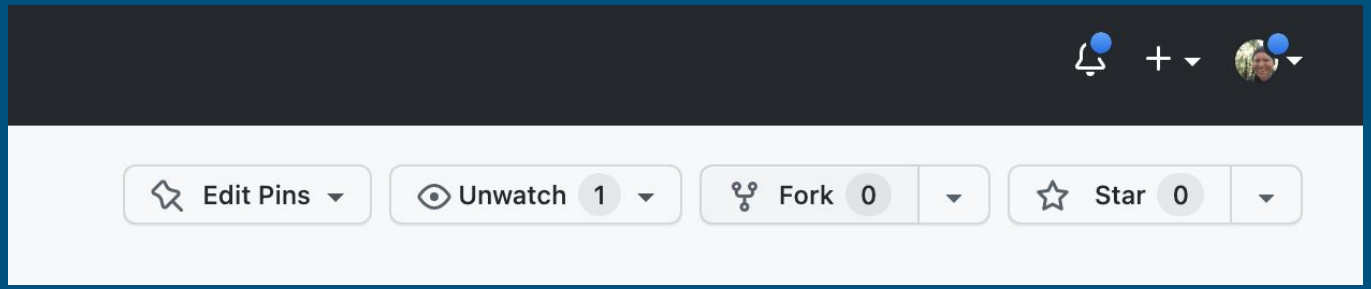


Create an account on GitHub



Fork my repository for this course

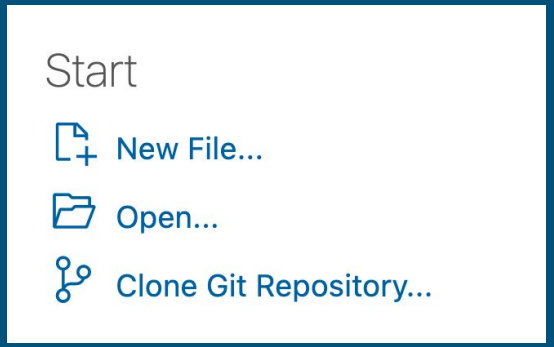
https://github.com/twin-bridges/pynet-ons-oct22

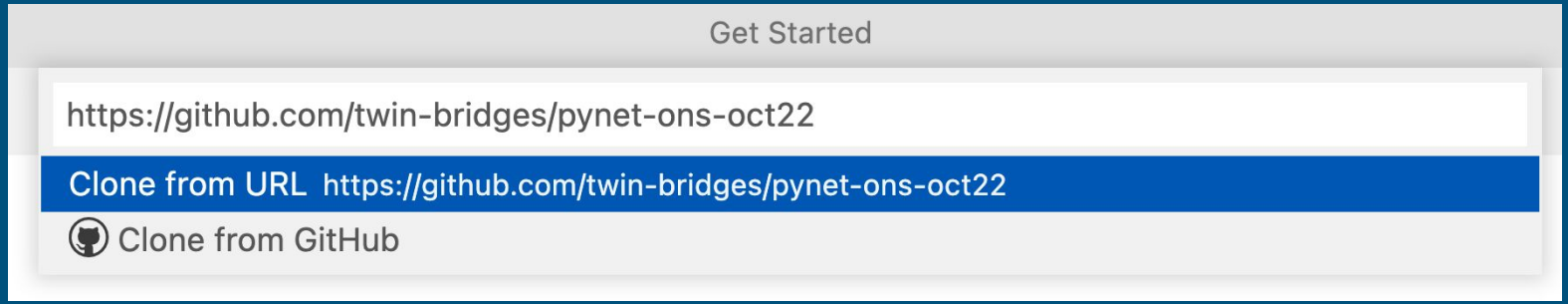


VS Code Setup

Clone the course's Git repository

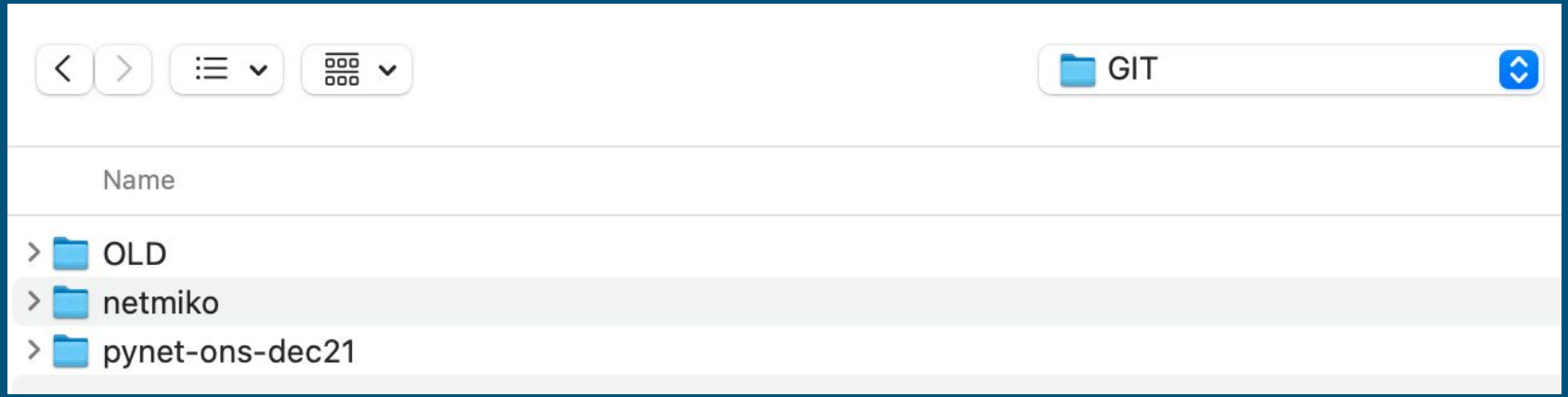
You must clone your fork of the repository.





VS Code Setup

Pick location for repository



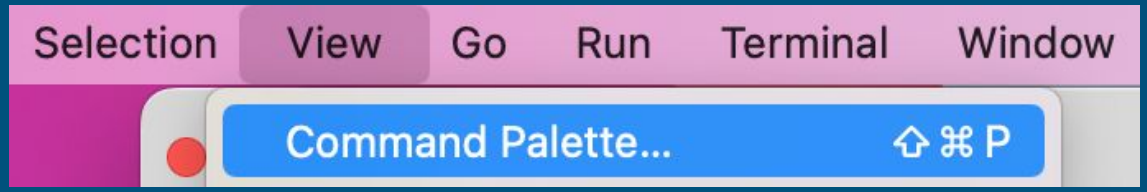
Create a Python Virtual Environment

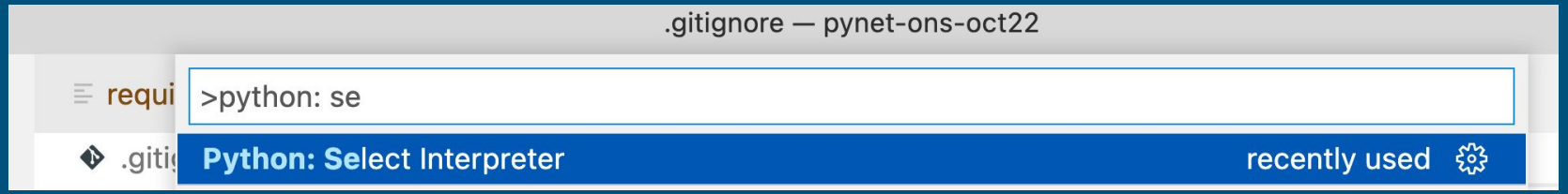
```
# Create the virtual environment
$ python3 -m venv .venv

# Activate the virtual environment
$ source .venv/bin/activate

$ which python3
/Users/ktbyers/GIT/pynet-ons-oct22/.venv/bin/
python3

# Deactivate the virtual environment
$ deactivate
```

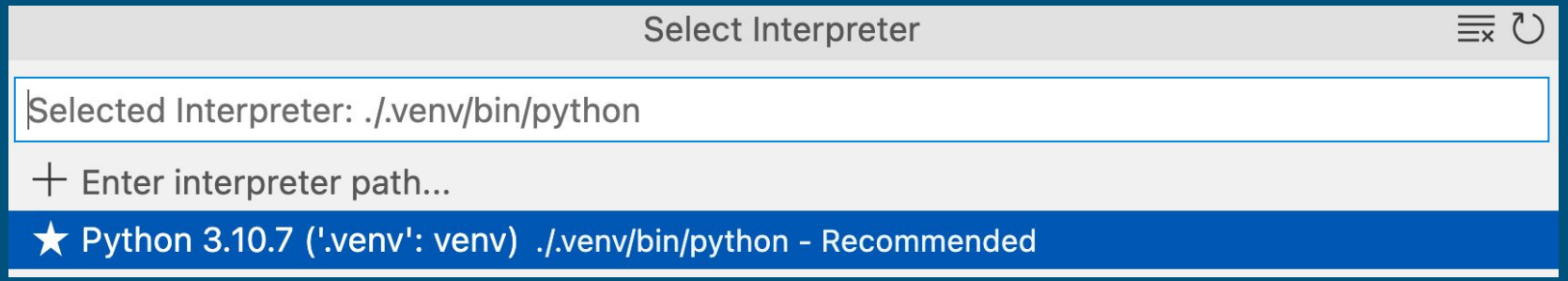
Install Dependencies in your Virtual Environment

```
$ pip list
Package    Version
---------- -------
pip        22.2.2
setuptools 63.2.0


$ pip install -r ./requirements-dev.txt

$ pip list
Package           Version
----------------- -------
appnope           0.1.3
asttokens         2.0.8
```

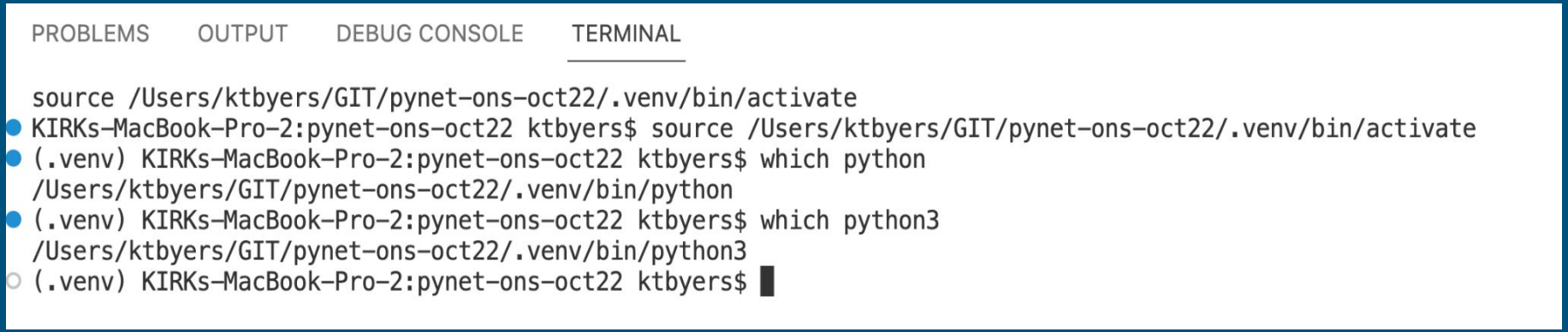
VS Code: Select the Virtual Environment Python







VS Code: Verify your Python

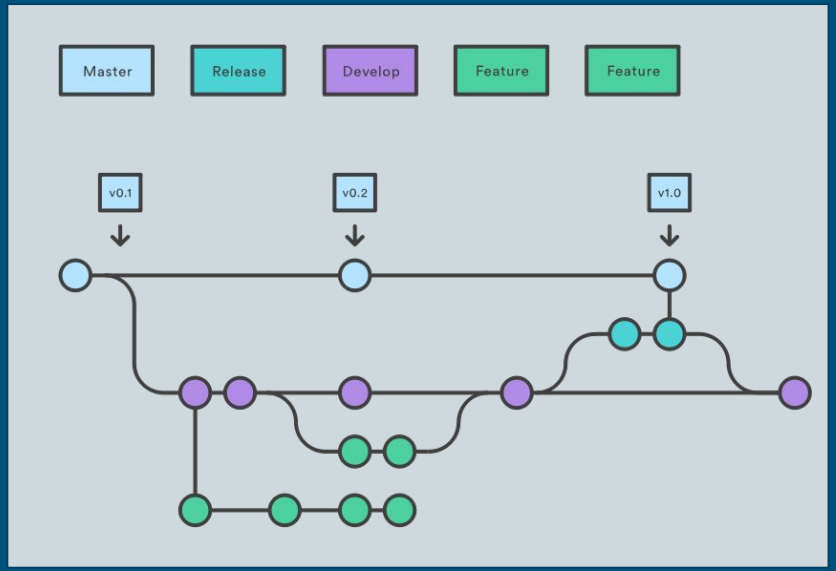


VS Code Settings File


```
{
    "python.linting.enabled": true,
    "python.formatting.provider": "black",
    "python.formatting.blackPath": "black",
    "python.linting.pycodestyleEnabled": true,
    "python.linting.pycodestylePath": "pycodestyle",
    "python.linting.pycodestyleArgs": [
        "--max-line-length=100"
    ],
    "editor.formatOnSave": true,
}
```

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*

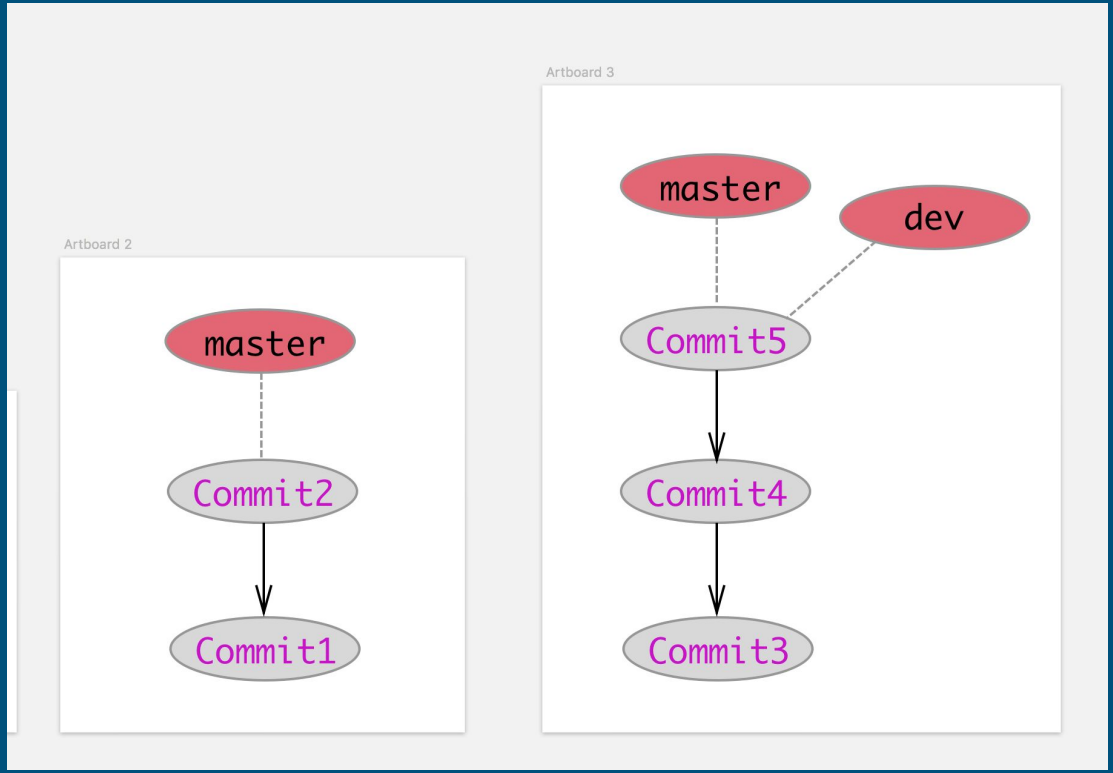
Git Push & Pull

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

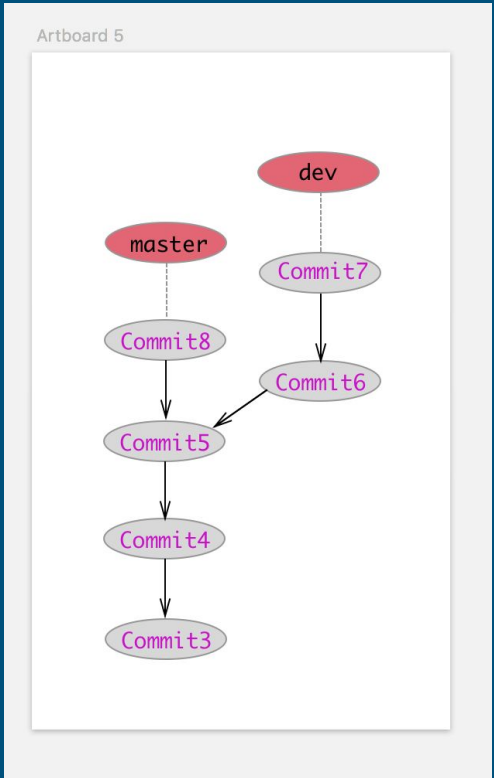
- git pull / git push
- git remote -v
- git remote add
- git branch -vv

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.

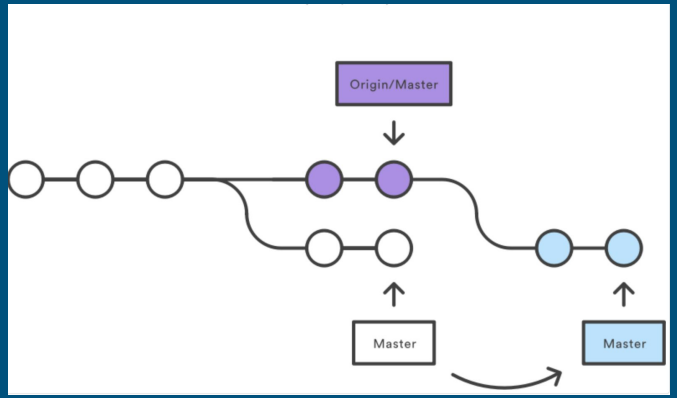
$ cat test2.py

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

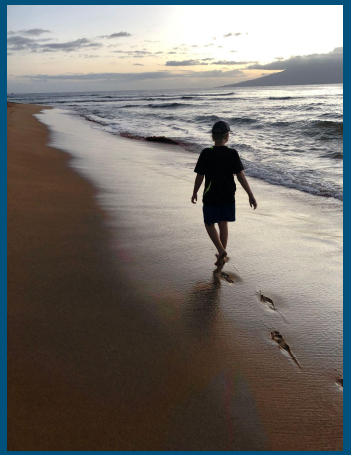


Git Exercises

Reference Commands:

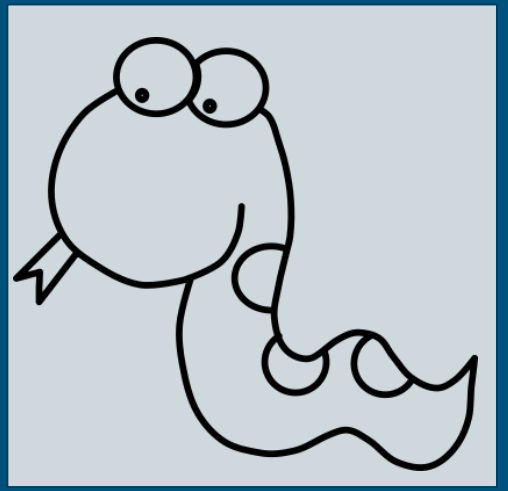
{{ github_repo }}/git_notes/git_commands.MD

Exercises:
./day1-virt/git/git_ex1.txt
./day1-virt/git/git_ex2.txt
./day1-virt/git/git_ex3.txt



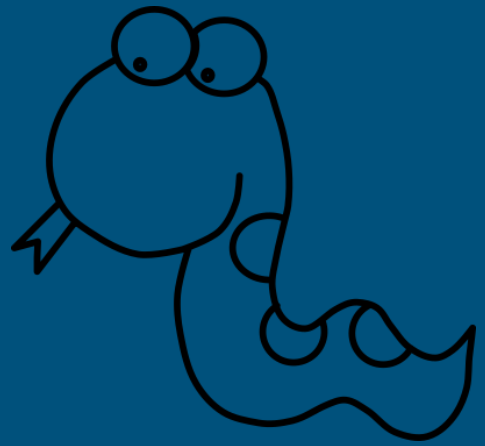
Why Python?

- 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



Python Characteristics

Indentation matters.

Use spaces not tabs.

Python programmers are particular.

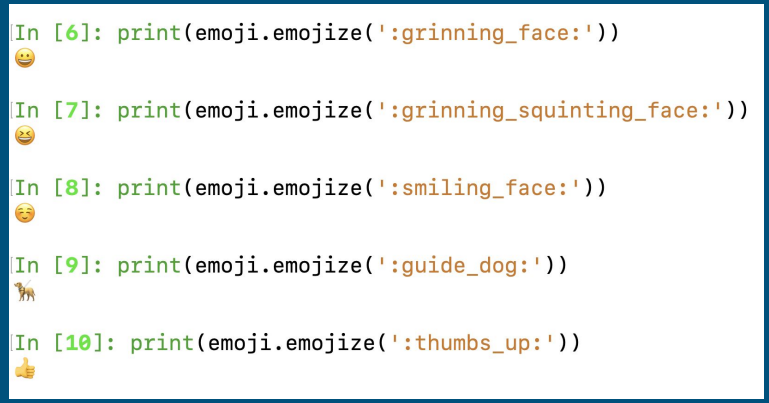
Python3

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



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

Numbers

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

Exercises:
./day1-virt/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
```

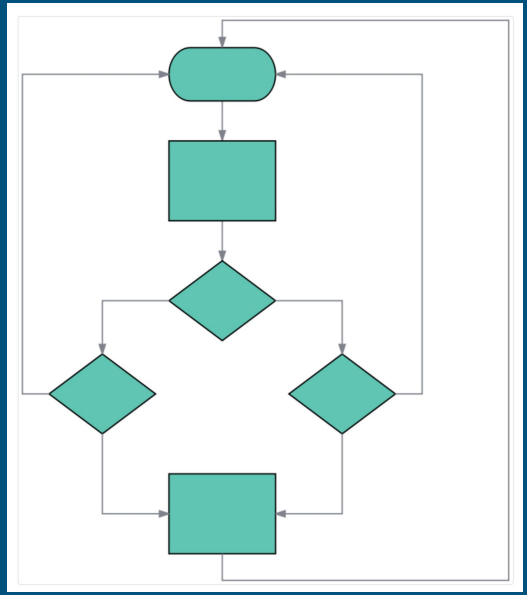
Exercises:
./day1-virt/py_lists/lists_ex1.txt
./day1-virt/py_lists/lists_ex2.txt

VS Code Debugging

- Setting breakpoints
- Step Over (next)
- Inspecting variables
- Using the Debug Console
- Step Into
- Step Out
- Adding Log Messages

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-virt/py_loops/loops_ex1.txt
./day1-virt/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-virt/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-virt/py_files/files_ex1.txt

Exception Handling

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

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

sys.path and $PYTHONPATH

```python
import sys
from rich import print

print(sys.path)
```

How does Python locate other Python Libraries?
Where does Python even look?

```
# Modify PYTHONPATH to get extra libraries
export PYTHONPATH=~/python-libs
export PYTHONPATH=$PYTHONPATH:~/DJANGOX/djproject/
```

```
>>> print(sys.path)
[
    '',
    '/Library/Frameworks/Python.framework/Versions/3.10/lib/python310.zip',
    '/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10',
    '/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/lib-dynload',
    '/Users/ktbyers/GIT/pynet-ons-oct22/.venv/lib/python3.10/site-packages'
]
>>>
```

$PYTHONPATH and VS Code

https://code.visualstudio.com/docs/python/environments#_use-of-the-pythonpath-variable

Use of the PYTHONPATH variable

The PYTHONPATH environment variable specifies additional locations where the Python interpreter should look for modules. In VS Code, PYTHONPATH can be set through the terminal settings (terminal.integrated.env.*) and/or within an `.env` file.

Libraries

```
import sys
from rich import print
from netmiko import ConnectHandler
```

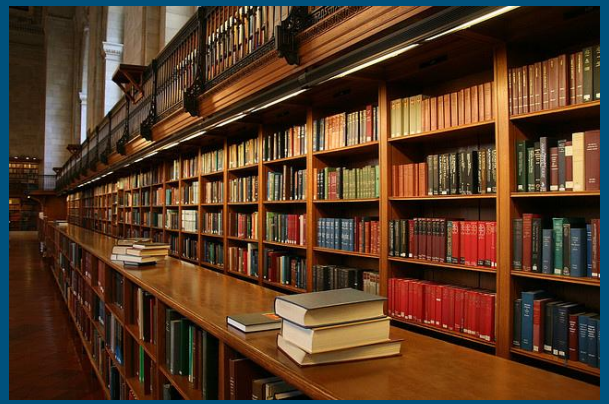


Photo: Viva Vivanista (Flickr)

PIP = Package Installer for Python

$ pip list

$ pip list | grep rich

$ pip uninstall rich

$ pip install rich==12.5.1

$ pip freeze

$ pip install -r ./requirements.txt

$ pip install -e .

pypi = Python Package Index

Python Linters

pylint or pycodestyle

Consistency and conventions make your life easier.

Finds obvious errors. Finds problems you might not be aware of.

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

Auto formatting with Python Black

Python Linters and VS Code

```
{
    "python.linting.enabled": true,
    "python.formatting.provider": "black",
    "python.formatting.blackPath": "black",
    "python.linting.pycodestyleEnabled": true,
    "python.linting.pycodestylePath": "pycodestyle",
    "python.linting.pycodestyleArgs": [
        "--max-line-length=100"
    ],
    "editor.formatOnSave": true,
}
```

Classes and Objects

```
class Server:
    def __init__(self, hostname, username, password):
        self.hostname = hostname
        self.username = username
        self.password = password

    def test_method(self):
        print(f"Device is: {self.hostname}")
        print(f"Username is: {self.username}")


svr1 = Server("test.domain.com", "admin", "passw")
svr1.test_method()
```

1. What is a class?
2. When would you want to create and use a class?
3. How do you create a class?
   a. What is the purpose of dunder-init?
   b. What is the meaning of “self”?
   c. How do you create object attributes?
4. How do you create methods?
5. How do you create objects (instances of the class)?
6. How do you call methods?

Exercises:
./day1-virt/py_classes/classes_ex1.txt
./day1-virt/py_classes/classes_ex2.txt

More Data Structures

Nesting Lists:

```
In [4]: print(my_list)
[
    ['rtr1', 'rtr2', 'sw1', 'sw2'],
    ['22.17.1.1', '22.17.1.2', '22.17.1.20', '22.17.1.21'],
    ['sj1', 'sj1', 'sj1', 'sj1']
]
```

```
In [6]: my_list[1]
Out[6]: ['22.17.1.1', '22.17.1.2', '22.17.1.20', '22.17.1.21']

In [7]: my_list[1][2]
Out[7]: '22.17.1.20'
```

Nesting Dictionaries

```
In [10]: print(net_devices)
{
    'rtr1': {'ip_addr': '22.17.1.1', 'vendor': 'cisco', 'platform': 'ios-xe'},
    'rtr2': {'ip_addr': '22.17.1.2', 'vendor': 'cisco', 'platform': 'ios-xe'},
    'sw1': {'ip_addr': '22.17.1.20', 'vendor': 'aruba', 'platform': 'aos8'},
    'sw2': {'ip_addr': '22.17.1.21', 'vendor': 'aruba', 'platform': 'aos8'}
}
```

```
In [11]: net_devices["sw1"]
Out[11]: {'ip_addr': '22.17.1.20', 'vendor': 'aruba', 'platform': 'aos8'}

In [12]: net_devices["sw1"]["vendor"]
Out[12]: 'aruba'
```

Nesting Dictionaries and Lists:

```
In [3]: print(data)
[{'protocol': 'O', 'type': 'E2', 'network': '0.0.0.0', 'mask': '0', 'distance': '110', 'metric': '1', 'n
exthop_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', 'distan
ce': '', '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': ''}]
```

```
In [4]: type(data)
Out[4]: list

In [5]: len(data)
Out[5]: 6
```

Nesting Dictionaries and Lists:

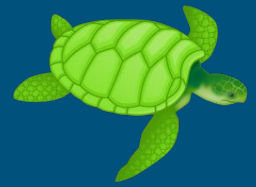
```
In [8]: data[0]
Out[8]:
{'protocol': 'O',
 'type': 'E2',
 'network': '0.0.0.0',
 'mask': '0',
 'distance': '110',
 'metric': '1',
 'nexthop_ip': '172.31.255.254',
 'nexthop_if': 'Vlan3967',
 'uptime': '3w6d'}
```

```
In [9]: data[0]['protocol']
Out[9]: 'O'

In [10]: data[0]['nexthop_ip']
Out[10]: '172.31.255.254'
```

Nesting Dictionaries and Lists:

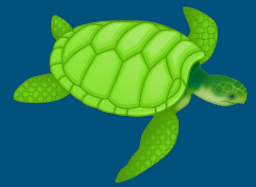
```
In [13]: for route_entry in data:
    ...:     print(f"Protocol: {route_entry['protocol']}")
    ...:
Protocol: O
Protocol: C
Protocol: L
Protocol: C
Protocol: C
Protocol: L
```

Data structures all the way down



```
In [5]: print(data)
{
    '_meta': {
        'int_vlan': {
            '_mappings': {
                'root': 'int_vlan',
                'key_list': {
                    'id': 'id',
                    'int_vlan_shut': 'int_vlan_shut',
                    'int_vlan_ip.ipparams': 'int_vlan_ip.ipparams',
                    'int_vlan_ip.ipaddr': 'int_vlan_ip.ipaddr',
                    'int_vlan_ip.ipmask': 'int_vlan_ip.ipmask',
                    'int_vlan_ip.dhcp-client': 'int_vlan_ip.dhcp-client',
                    'int_vlan_ip.client-id': 'int_vlan_ip.client-id',
                    'int_vlan_ip.cid': 'int_vlan_ip.cid',
```

Step down into the data structure: Layer-by-layer



```
In [16]: data.keys()
Out[16]: dict_keys(['_meta', '_data'])

In [17]: type(data["_data"])
Out[17]: dict

In [18]: data["_data"].keys()
Out[18]: dict_keys(['int_vlan'])
```

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-virt/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 rich.print to print these two data structures to the screen.

Exercises:
./day1-virt/exercise_review/review_ex2.txt

Review Exercise

Exercises:
./day1-virt/complex_data_struct/struct_ex1.txt

```
In [2]: import json

In [3]: with open("struct_data1.json") as f:
   ...:     data = json.load(f)
   ...:
```