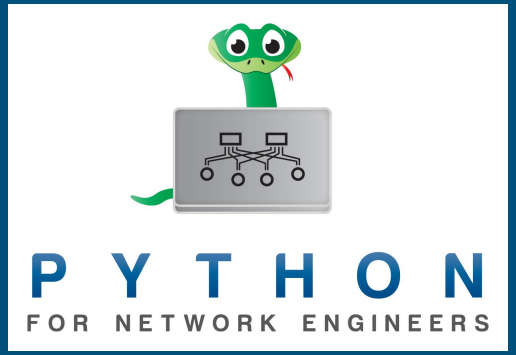
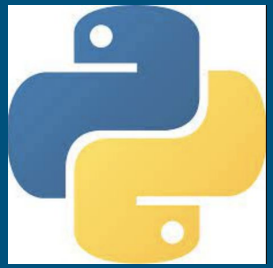
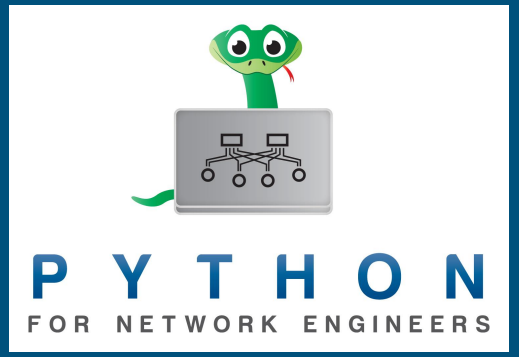




Virtual Training Session
October 2022 (Day2)

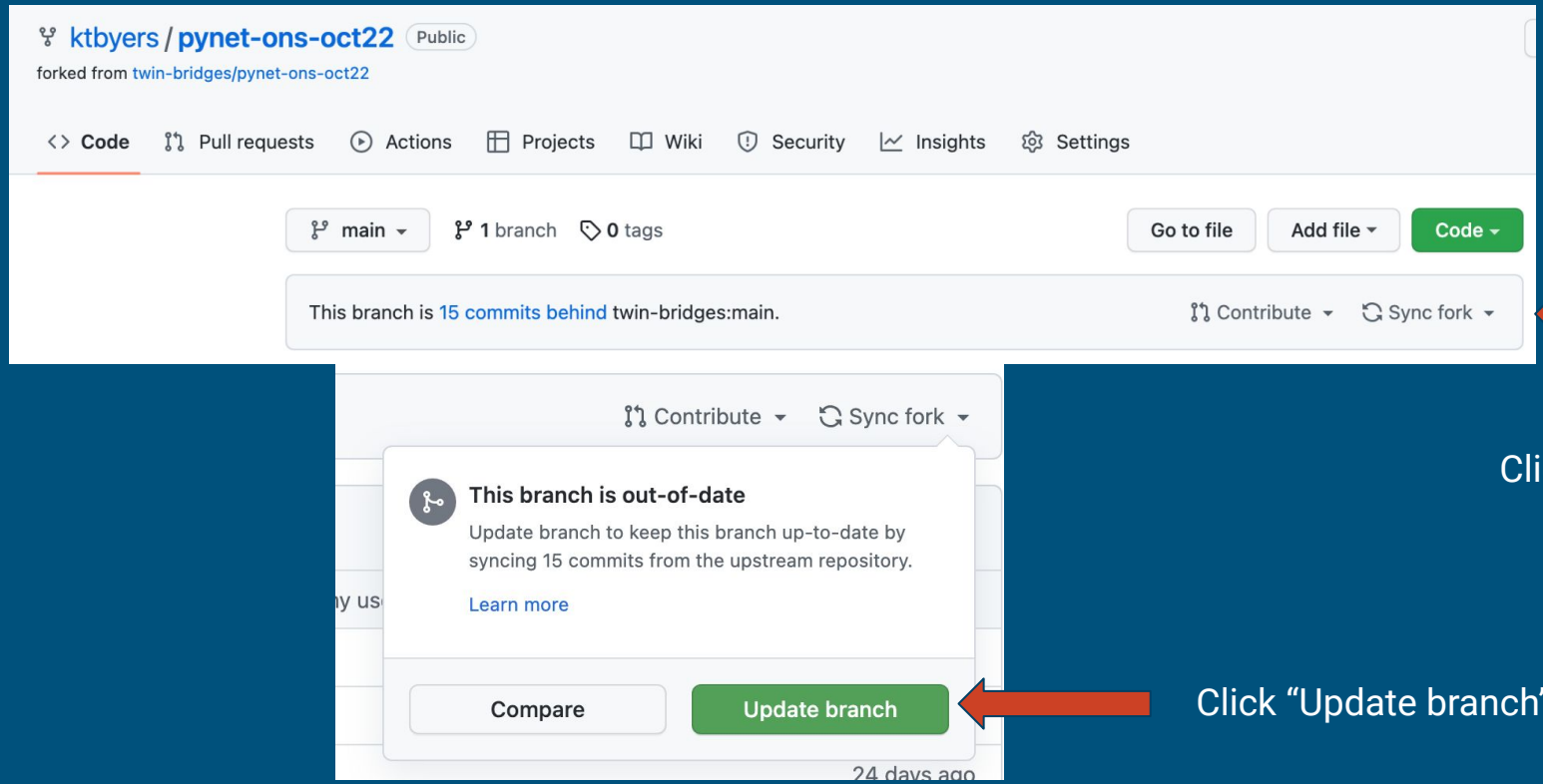
Python Fundamentals and Corresponding Examples & Exercises



- Strings
- Numbers
- Lists
- Conditionals
- Loops
- Dictionaries
- Files

Rebase GitHub Project

Your fork of the course repository (not mine)

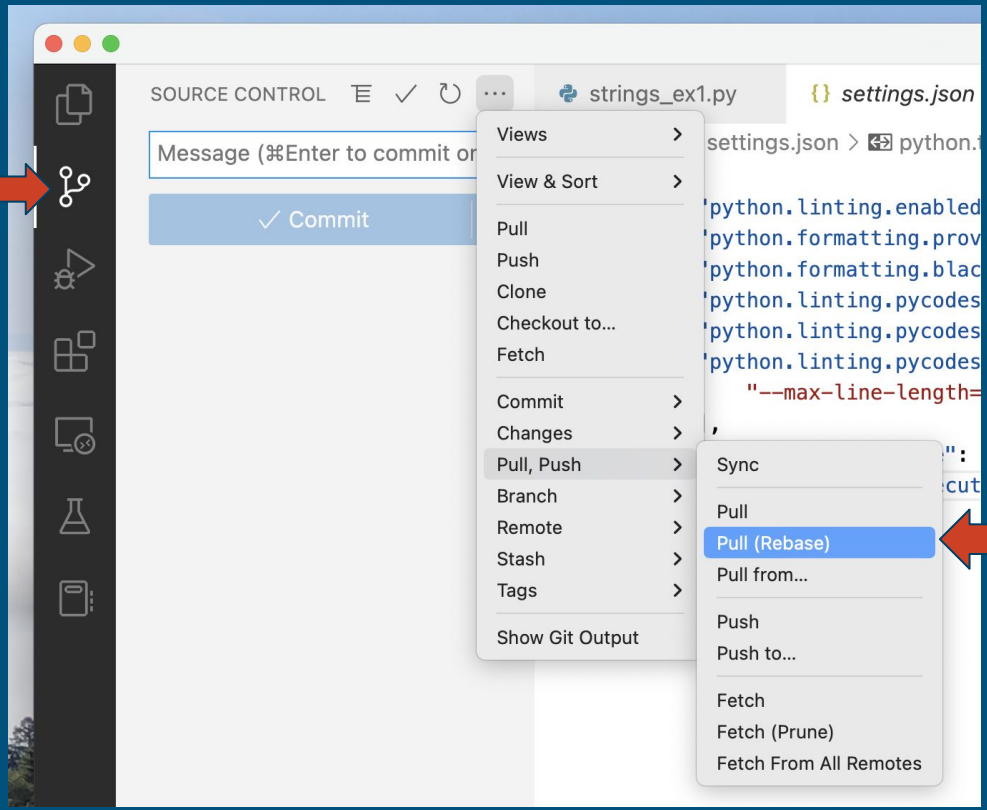


Click “Sync Fork”

Click “Update branch”

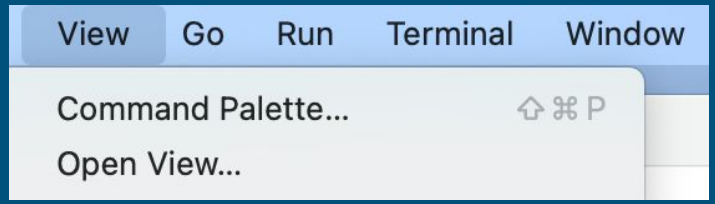
You need to pull the changes into your computer

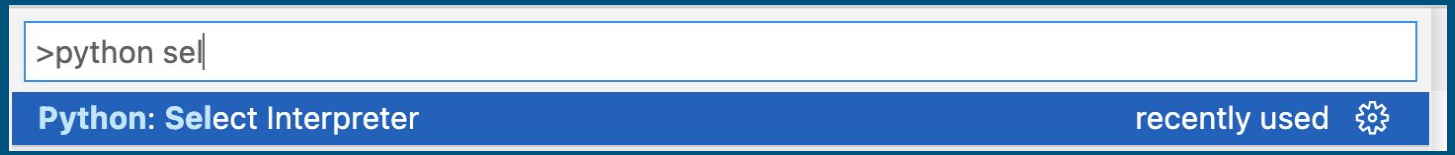
Git Options

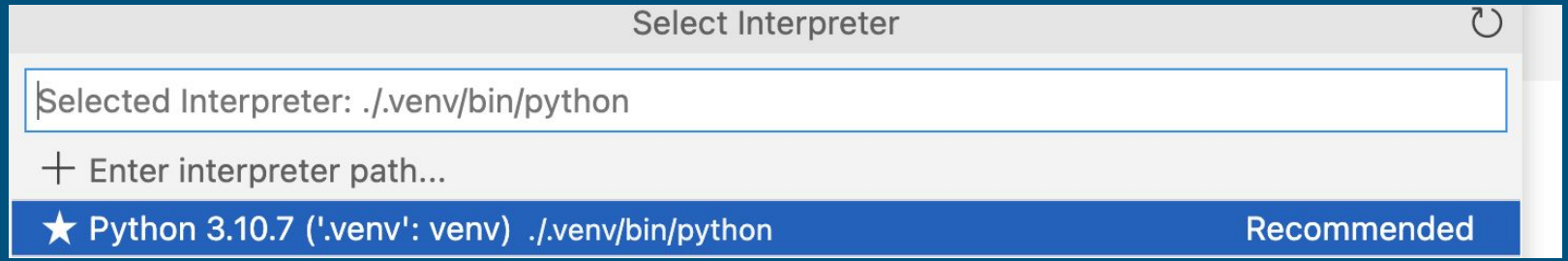


Pull (Rebase)

You might need to reselect your Python Interpreter







Virtual Environment Active

VS Code > Terminal > New Terminal

macOS: source .venv/bin/activate

Windows: ./.venv/scripts/activate

Verify Python:

python -m pip list [macOS / Windows]

Python Characteristics

Indentation matters.

Use spaces not tabs.

Python programmers are particular.

Python3

flickr: @eekim

Review

Executing a Python program

Review

From VS Code:

Pick the file > upper right, select “run triangle”

From Terminal:

(.venv) $ python simple.py

Hello

Review Exercise:

Exercises:
./day2/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.

Some Parts of This: Strings

```
_, serial_number = line.split("Processor board ID")
```

What does .split() return?

```
In [5]: line
Out[5]: 'This is a line of text'

In [6]: line.split("This is")
Out[6]: ['', ' a line of text']
```

What does .split() return?

```
In [7]: line.split(" ")
Out[7]: ['This', 'is', 'a', 'line', 'of', 'text']
```

Returns a list.

```
In [12]: line = "This is a line of   text"

In [13]: line.split()
Out[13]: ['This', 'is', 'a', 'line', 'of', 'text']

In [14]: line.split(" ")
Out[14]: ['This', 'is', 'a', 'line', 'of', '', '', 'text']
```

What the heck happens here: Unpacking

```
_, serial_number = line.split("Processor board ID")
```

```
>>> line
'Processor board ID FTX1512038X'
>>> line.split("Processor board ID")
['', ' FTX1512038X']
```

line.split returns a two-element List

```
>>> var1, var2 = line.split("Processor board ID")
>>> var1
''
>>> var2
' FTX1512038X'
```

Immediately unpack into two variables

Some Parts of This: f-strings

```
print(f"\nSerial Number is {serial_number}\n")
```

String interpolation: embedding variables in strings

VS Code and Execution Location

```
 9      ],
10      "editor.formatOnSave": true,
11+     "python.terminal.executeInFileDir": true
12  }
```

VS Code file location issue

VS Code and Debug Location

```
{
    // Use IntelliSense to learn about possible attributes.
    // Hover to view descriptions of existing attributes.
    // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
    "version": "0.2.0",
    "configurations": [
        {
            "name": "Python: Current File",
            "type": "python",
            "request": "launch",
            "program": "${file}",
            "console": "integratedTerminal",
            "justMyCode": true,
            "cwd": "${fileDirname}"
        }
    ]
}
```

Create a launch.json file in .vscode folder

Specify

VS Code file location issue

Strings

Exercises:
./day2/py_strings/strings_ex1.txt

Read in the “aruba_show_ap_database.txt” file.

Process the data such that all of the header and footer information is excluded.

In other words, only print out the tabular data from the file.
Your output should look as follows:

```
library-1       sjc     225     10.10.10.13     Up 8m:29s       Rc2     10.5.200.21     0.0.0.0
library-1       sjc     225     10.10.10.10     Down            Rc2     10.5.200.21     0.0.0.0
library-2       sjc     225     10.10.10.12     Down            Rc2     10.5.200.21     0.0.0.0
rm135-1         sjc     135     10.10.10.9      Down            Rc2     10.5.200.21     0.0.0.0
rm135-2         sjc     135     10.10.10.11     Down            Rc2     10.5.200.21     0.0.0.0
rm137-1         sjc     225     192.168.1.1     Up 1h:2m:05s    R       10.5.200.21     0.0.0.0
rm137-2         sjc     225     10.10.10.8      Down            Rc2     10.5.200.21     0.0.0.0
```

Strings and Dictionaries

Expand on exercise 1, except now process the tabular data to extract the AP name, the AP IP address, and the AP status.

Create a new dictionary where the key is the AP name and the corresponding value is the AP status.

Normalize both the AP name and the AP status to be all lower case.

Use rich to print out this new dictionary.

Exercises:
./day2/py_strings/strings_ex2.txt

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()
```

Exercise

Exercises:
./day2/py_files/files_ex1.txt

But what about lists?

Review

Exercises:
./day2/py_lists/files_ex1.txt

a. Create a list with five strings

b. Use append to add two strings to the list

c. Use pop to remove the first element

d. Find the length of the list

e. Sort the list

JavaScript and Ruby refers to this data structure as an Array

f. Loop over the list and print both the index and the value. For printing use f-strings and make the index a 3-character width field; the value should use a 15-character width field (right aligned).

How to access list indexes?


```
In [5]: my_list = ["It", "was", "the", "best", "of"]

In [6]: my_list[3]
Out[6]: 'best'
```

Review

Zero-based indices

```
In [7]: my_string = "This is some sentence"

In [8]: len(my_string)
Out[8]: 21

In [9]: my_string[0]
Out[9]: 'T'
```

Both lists and strings are sequences. They have an inher order and you can access the using indices.

List Exercise

a. Read the file "show_version.txt" (in the current directory). Remember the VS Code file location issue.

b. Read in the contents of the file using the readlines() method. This will create a list.

c. Print the length of the list. This will give you the number of lines in that file.

d. Print out line number 1 and line number 35 respectively. Remember indices are zero-based.

e. Split line number 1 into fields using a comma as the field separator. One of these fields should be the OS Version. Print this to standard output.

Exercises:
./day2/py_lists/files_ex2.txt

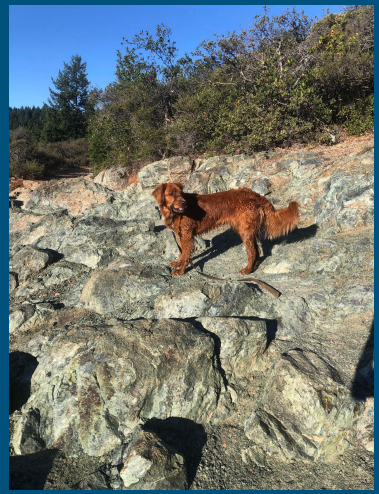
Another Example Exercise

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

1. Create a dictionary where the keys are the interface names

2. The corresponding values should be the “Protocol” field.

3. Use rich.print to print out your data structure.



Exercises:
./day2/review_exercises/review_ex1.txt

How about those booleans?

```
In [2]: print(line)
Interface                  IP-Address      OK? Method Status                Protocol

In [3]: "Interface" in line
Out[3]: True
```

Membership check

Boolean Result

```
In [4]: "Interface" in line and "Protocol" in line
Out[4]: True
```

Boolean Logic: and, or, not

And Dictionaries?

Review

Blank dictionary: curly braces

Adding a key-value pair

```
In [9]: my_ds = {}

In [10]: my_ds['FastEthernet0'] = 'up'

In [11]: my_ds
Out[11]: {'FastEthernet0': 'up'}
```

Looping over Dictionaries?

Review

```
In [1]: my_ds
Out[1]:
{'FastEthernet0': 'down',
 'FastEthernet1': 'down',
 'FastEthernet2': 'down',
 'FastEthernet3': 'down',
 'FastEthernet4': 'up',
 'Vlan1': 'down'}

In [2]: for k, v in my_ds.items():
   ...:     print(f"{k} -> {v}")
   ...:
FastEthernet0 -> down
FastEthernet1 -> down
FastEthernet2 -> down
FastEthernet3 -> down
FastEthernet4 -> up
Vlan1 -> down
```

.items() returns both the key and the value

Dictionary 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:
./day2/review_exercises/review_ex2.txt

Another Example Exercise

Read the ‘show_ip_bgp.txt’ file.

Strip out the BGP header information so you are just left with the route table.

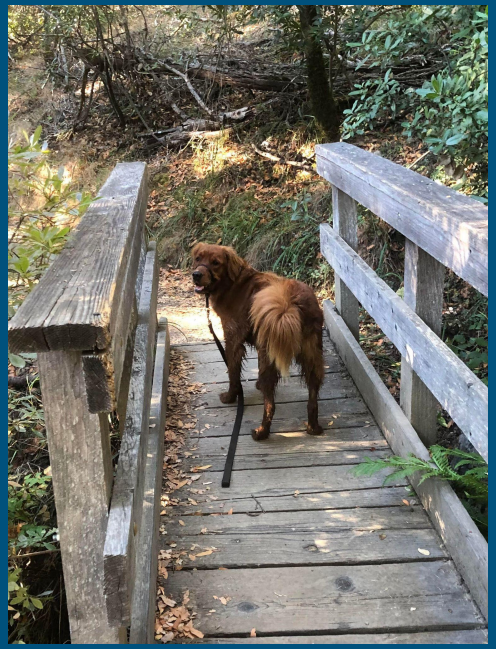
Parse each BGP line such that you retrieve the prefix and the as_path.

Save the prefix and as_path to a file.

Exercises:
./day2/review_exercises/review_ex2.txt

A bit loopy

Exercise



You can record a timestamp using Python's time library:

```
import time
now = time.time()
```

You can then track elapsed time as:

```
elapsed_time = time.time() - now
```

Finally, you can sleep a number of seconds using:

```
time.sleep(sleep_time)
```

A bit loopy

Exercise

1. Take a timestamp using time.time().

2. Enter into a while loop. Stay in this while loop until 15 seconds have passed (relative to your timestamp).

3. Inside the while loop, sleep for 1 second. Additionally, print out the elapsed time since your timestamp.

4. You should exit your while loop after 15 seconds have passed.

5. Outside of the while loop. Print the total elapsed time (one last time).

Exercises:
./day2/py_loops/loops_ex2.txt

Loops

- for
- while
- break
- continue
- enumerate

Remember your syntax

Remember break and continue

Remember enumerate

Strings

Remember some of your string methods

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

Membership

Escaping special characters

String interpolation

python™

Numbers

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

```
In [1]: var1 = 22.7

In [2]: type(var1)
Out[2]: float

In [3]: round(var1)
Out[3]: 23
```

```
In [4]: import math

In [5]: math.pi
Out[5]: 3.141592653589793

In [6]: var2 = math.pi

In [7]: round(var2, 2)
Out[7]: 3.14
```

```
In [8]: var3 = "88"

In [9]: type(var3)
Out[9]: str

In [10]: int(var3)
Out[10]: 88
```

Conditionals

Remember your syntax

```
a = 22
if a == 15:
    print("Hello")
elif a >= 7:
    print("Something")
else:
    print("Nada")
```

Expressions evaluate to True or False

Final catchall

Exercise

This ends up being challenging

1. Prompt a user to enter an IP address.

2. Check that the IP address has four octets and that each octet ranges between 0 and 255. If the check fails, re-prompt the user for a valid IP address.

3. Continue prompting until a valid IP address is returned (using the very simple IP check specified above).

4. Print out the IP address that was entered.

5. Test that your code works properly.

Exercises:
./day2/eod_exercises/exercise1.txt

Exercise

1. Read the contents of "show_vlan.txt". This is from an Arista vEOS switch.

2. Using either readlines() or splitlines() loop over the contents of this file. Skip the header information.

3. Create a new dictionary where the key is the vlan_id. The corresponding value should be a dictionary containing the
following key-value pairs: vlan_name, vlan_status, and ports. The ports key should refer to a list of ports.

4. Use rich.print to print out your final data structure. Your final data structure should be similar to the following:

```
{
    '1': {'vlan_name': 'default', 'vlan_status': 'active', 'ports': ['Cpu,', 'Et1']},
    '2': {'vlan_name': 'VLAN0002', 'vlan_status': 'active', 'ports': ['Et2']},
    '3': {'vlan_name': 'VLAN0003', 'vlan_status': 'active', 'ports': ['Et3']},
    '4': {'vlan_name': 'VLAN0004', 'vlan_status': 'active', 'ports': ['Et4']},
    '5': {'vlan_name': 'VLAN0005', 'vlan_status': 'active', 'ports': ['Et5']},
    '6': {'vlan_name': 'VLAN0006', 'vlan_status': 'active', 'ports': ['Et6']},
    '7': {'vlan_name': 'VLAN0007', 'vlan_status': 'active', 'ports': ['Et7']}
}
```

Exercises:
./day2/eod_exercises/exercise2.txt