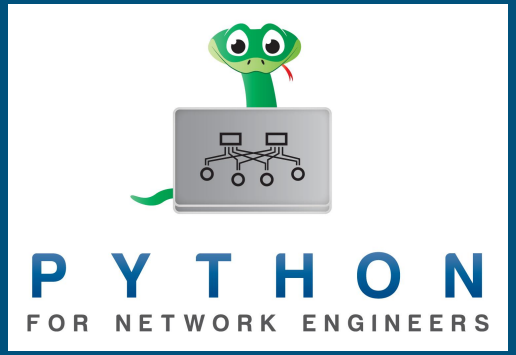
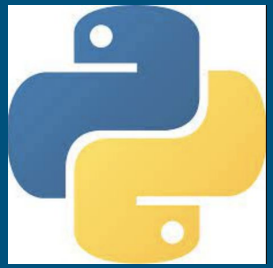




Python Automation Training Session
Dec 2022

$ whoami

Kirk Byers
Network Engineer
CCIE #6243 (emeritus)

Programmer
Netmiko
NAPALM
Nornir

Teach Python, Ansible, Nornir in a Network Automation context

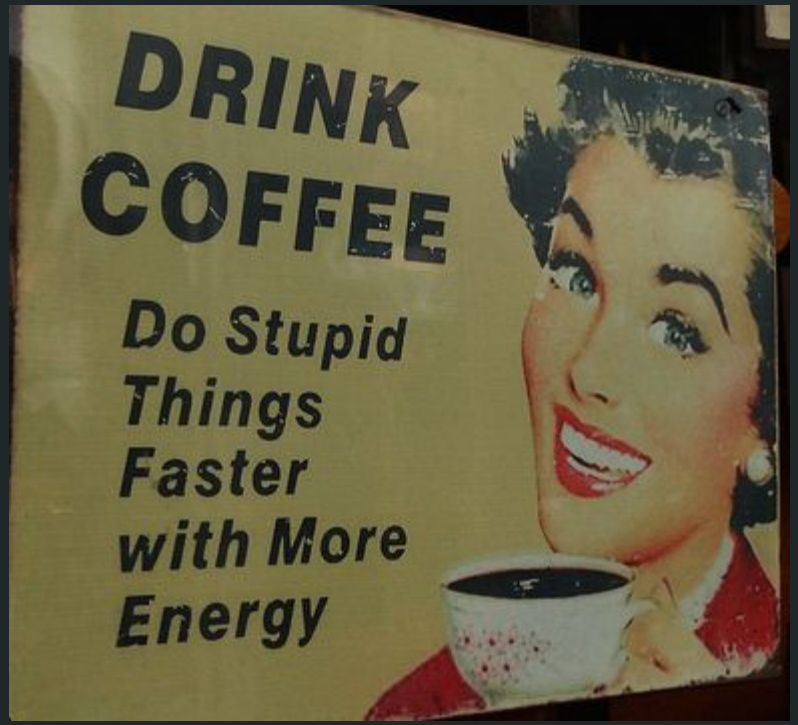
General:

Virtual-Training Day
Dec 2nd, Day1 (Fri) / 9AM - 4:30PM Central

Dec 12th, Day2 (Mon)

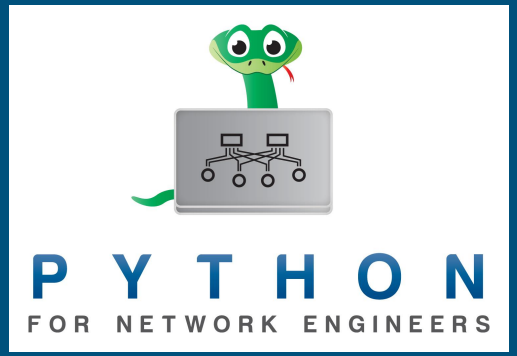
Focused/Minimize Distractions

The exercises are important.



Flickr: Ben Sutherland

Day1 Training Schedule



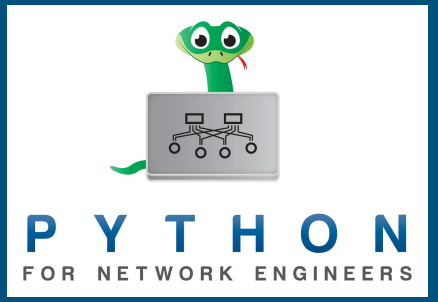
Course Introduction

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

VS Code Setup

Git Review

Day1 Training Schedule



Why Python?

Python Fundamentals

- *Strings/Numbers*
- *Lists*
- *Conditionals*
- *Loops*

Environment Setup (macOS)

Things that should be working at this point:

```
$ python3
Python 3.10.8 (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 --list
user.email=ktbyers@twb-tech.com
user.name=Kirk Byers
```

Environment Setup (Windows)

Things that should be working at this point:

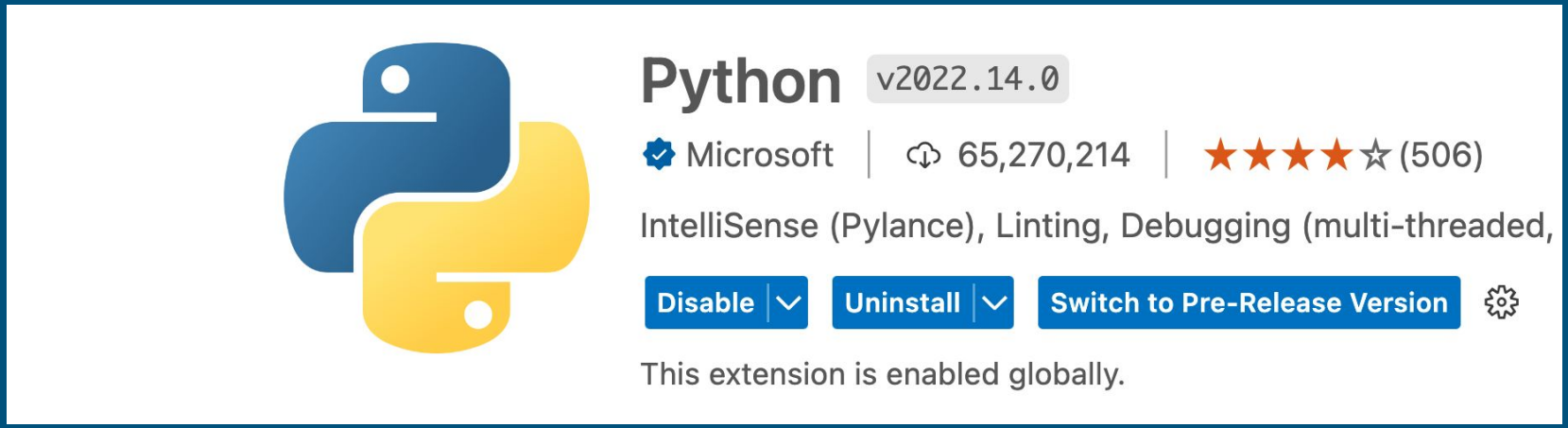
```
C:\Users\Administrator>py
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> quit()
```

```
C:\Users\Administrator>git version
git version 2.38.1.windows.1
```

```
C:\Users\Administrator>git config --global --list
user.name=Kirk Byers
user.email=ktbyers@twb-tech.com
```

VS Code Setup

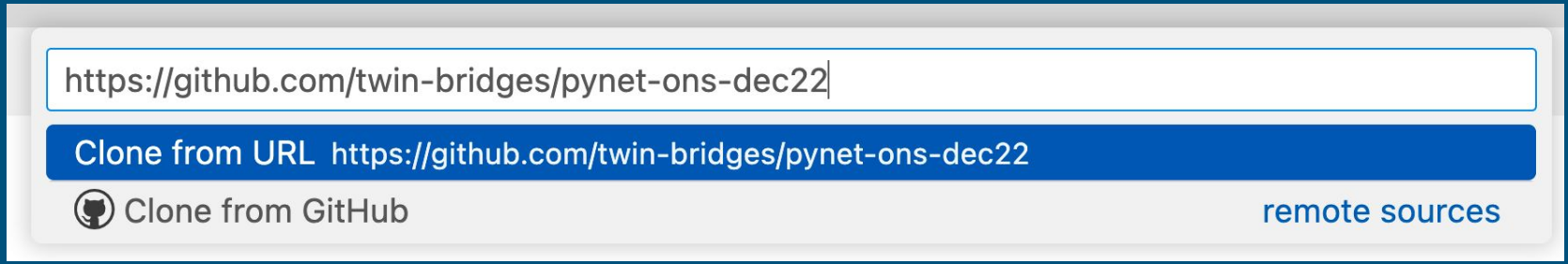
Install Python extension for VS Code



VS Code Setup

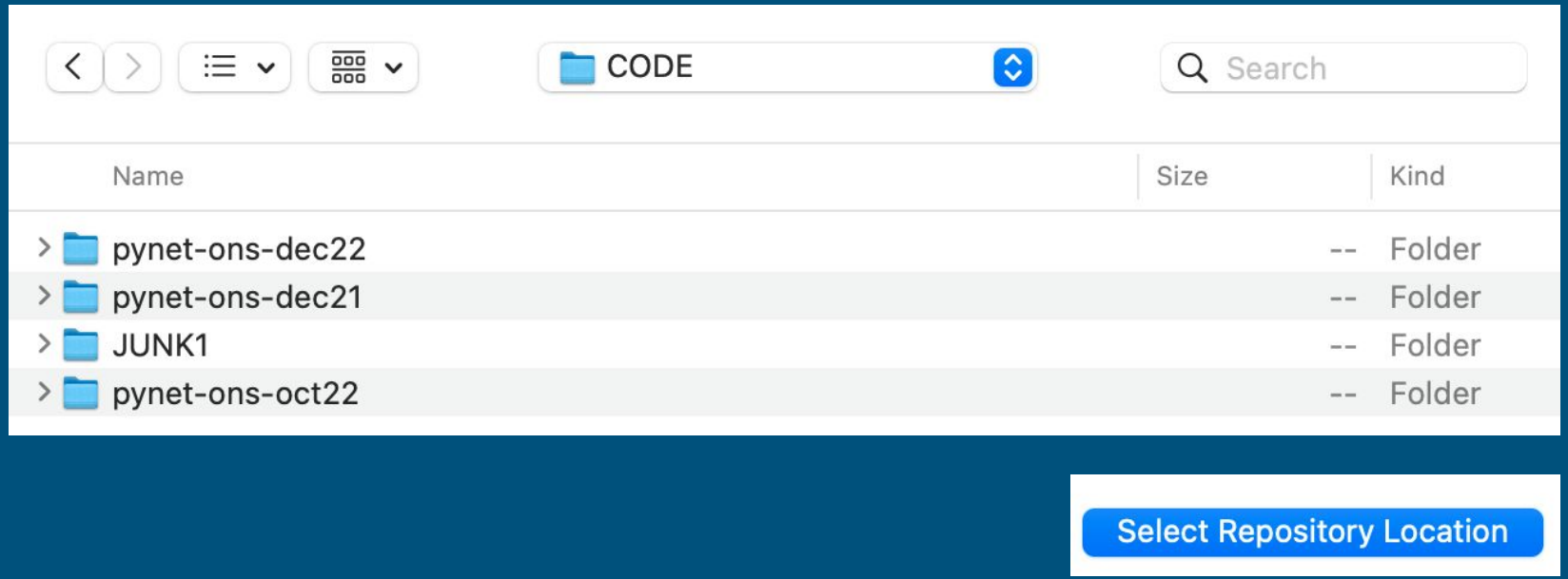
PYTHON FOR NETWORK ENGINEERS

Clone the course's Git repository.



VS Code Setup

Pick a location for the repository



Create a Python Virtual Environment (macOS)

```
# 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
```

Create a Python Virtual Environment (Windows)

'cd' to your repository directory

```
c:\>cd c:\Users\Administrator\CODE\pynet-ons-dec22

c:\Users\Administrator\CODE\pynet-ons-dec22>
```

Create the virtual environment.

```
c:\Users\Administrator\CODE\pynet-ons-dec22>py -m venv .venv

c:\Users\Administrator\CODE\pynet-ons-dec22>dir
 Volume in drive C has no label.
 Volume Serial Number is 8636-CA6C

 Directory of c:\Users\Administrator\CODE\pynet-ons-dec22

11/29/2022  07:31 PM    <DIR>          .
11/29/2022  07:16 PM    <DIR>          ..
11/29/2022  07:16 PM             1,928 .gitignore
11/29/2022  07:31 PM    <DIR>          .venv
11/29/2022  07:16 PM            11,558 LICENSE
11/29/2022  07:16 PM                17 README.md
               3 File(s)         13,503 bytes
               3 Dir(s)  14,655,033,344 bytes free
```

Create a Python Virtual Environment (Windows)

Activate the virtual environment.

```
c:\Users\Administrator\CODE\pynet-ons-dec22>.\.venv\Scripts\activate

(.venv) c:\Users\Administrator\CODE\pynet-ons-dec22>
```

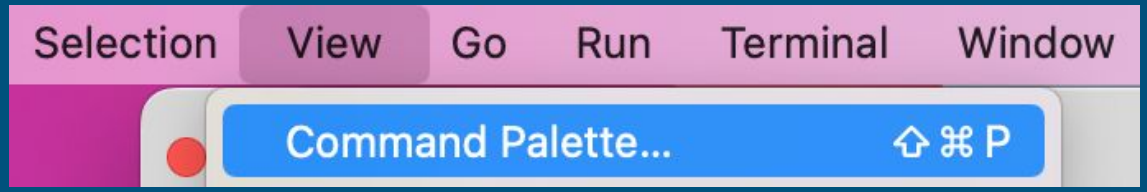
Prompt changes

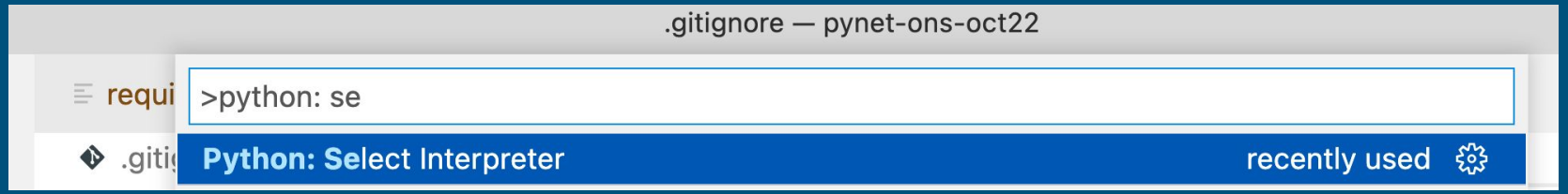
Install Dependencies in your Virtual Environment

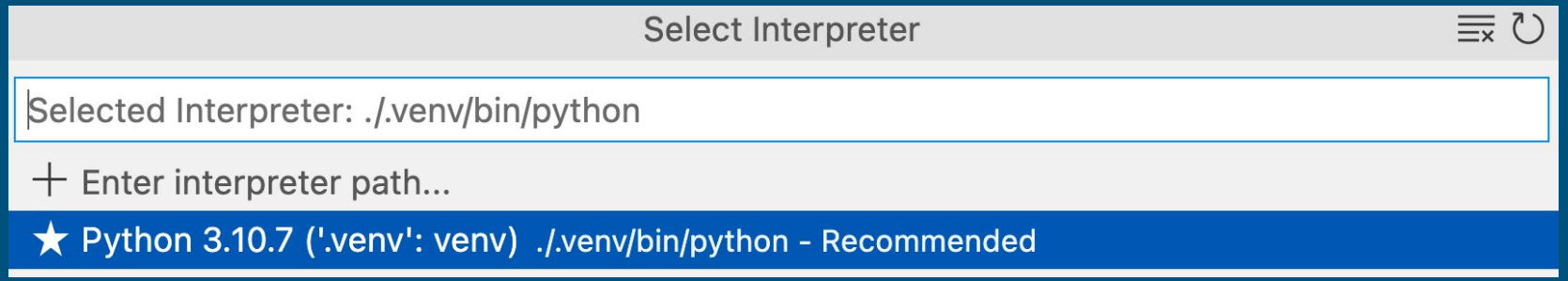
```
(.venv) PS C:\Users\Administrator\CODE\pynet-ons-dec22> py -m pip list
Package    Version
---------- -------
pip        22.3
setuptools 65.5.0

[notice] A new release of pip available: 22.3 -> 22.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
(.venv) PS C:\Users\Administrator\CODE\pynet-ons-dec22> py -m pip install -r .\requirements-dev.txt
Collecting ipython
  Downloading ipython-8.7.0-py3-none-any.whl (761 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 761.7/761.7 kB 4.4 MB/s eta 0:00:00
```

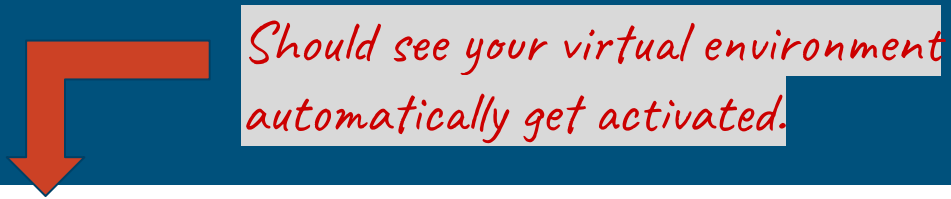
VS Code: Select the Virtual Environment Python







VS Code: Verify your Python



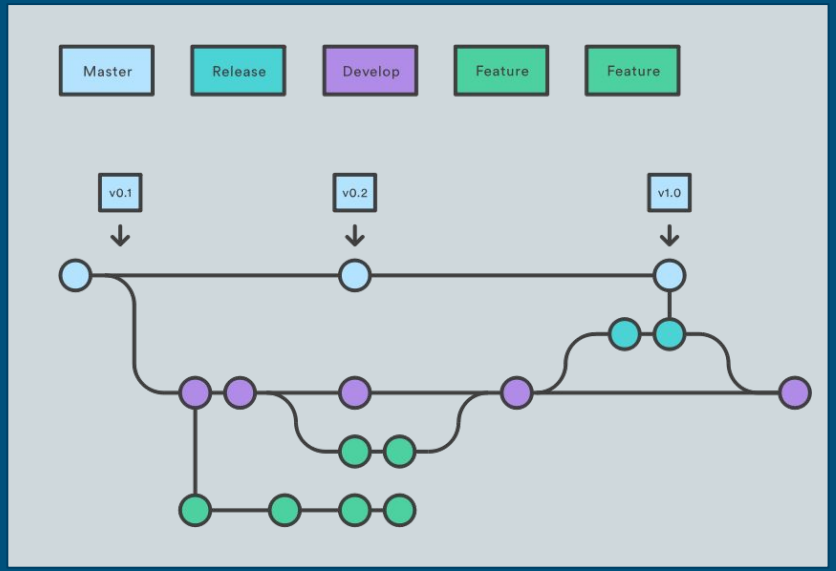

```
PS C:\Users\Administrator\CODE\pynet-ons-dec22> & c:/Users/Administrator/CODE/pynet-ons-dec22/.venv/Scripts/Activate.ps1
(.venv) PS C:\Users\Administrator\CODE\pynet-ons-dec22>
```


```
(.venv) PS C:\Users\Administrator\CODE\pynet-ons-dec22> python -m IPython
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.7.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
```

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, git, git out of here.

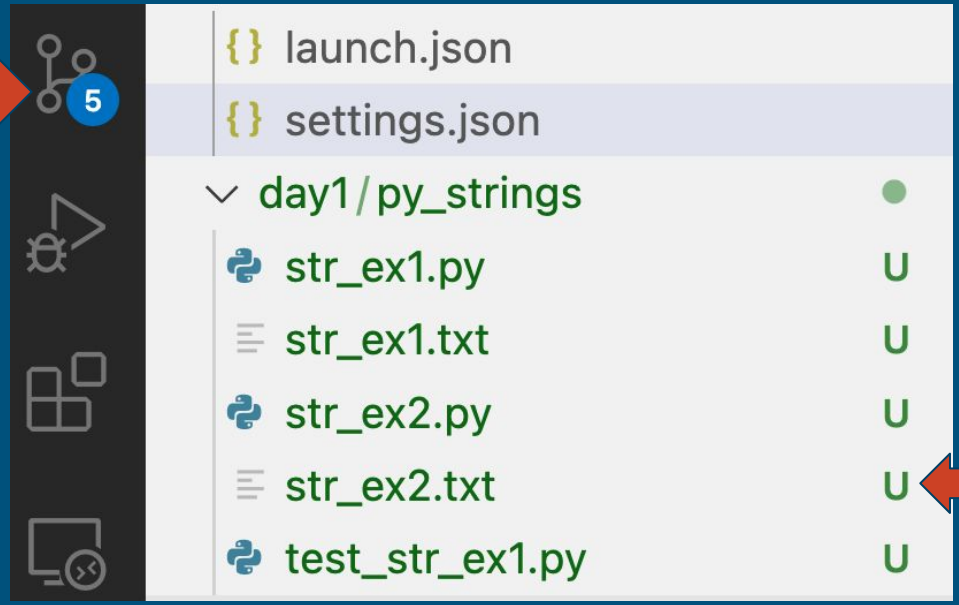


- Why care about Git?
- Git and GitHub
- Some principles of how Git works
  - Tracking files and directories across time
  - All objects are stored in the .git directory
  - You can swap your working set of files
  - Distributed
- Creating a repository on GitHub
- Cloning a repository
- git init
- Files have four different states: untracked, modified, staged, committed

Git Status

What is the current state of the files?

VS Code is saying there are 5 files that are showing up in 'git status'

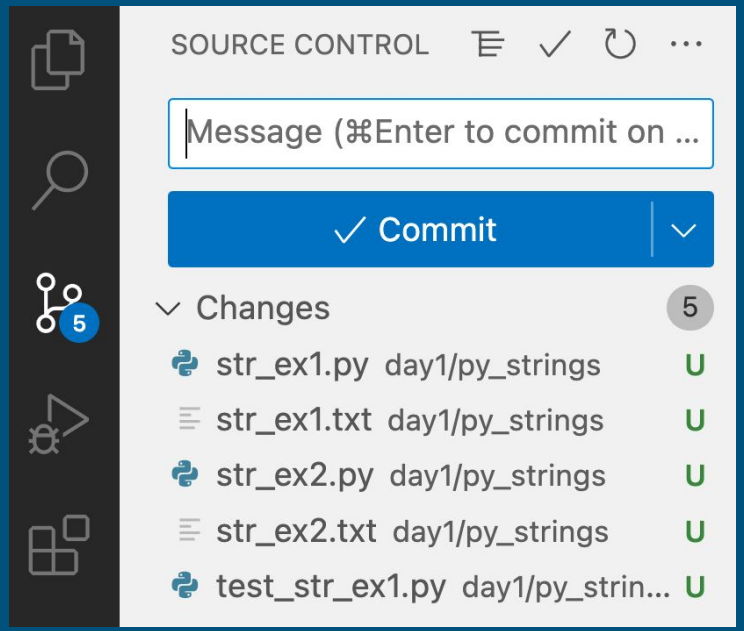


U indicates they are untracked.

Git Status

What is the current state of the files?

Click on "Git" icon and shows you a different view on the changes.



Git Status

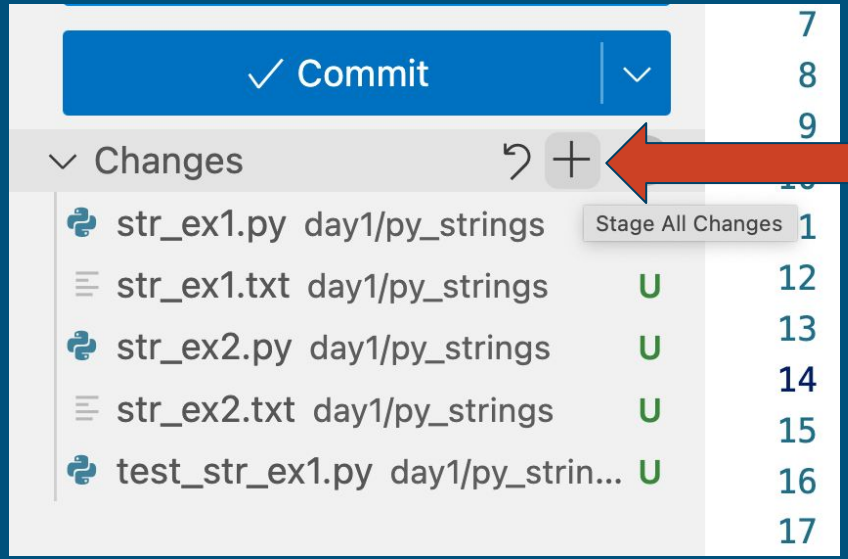
What is the current state of the files?


```
(.venv) $ git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        day1/
```

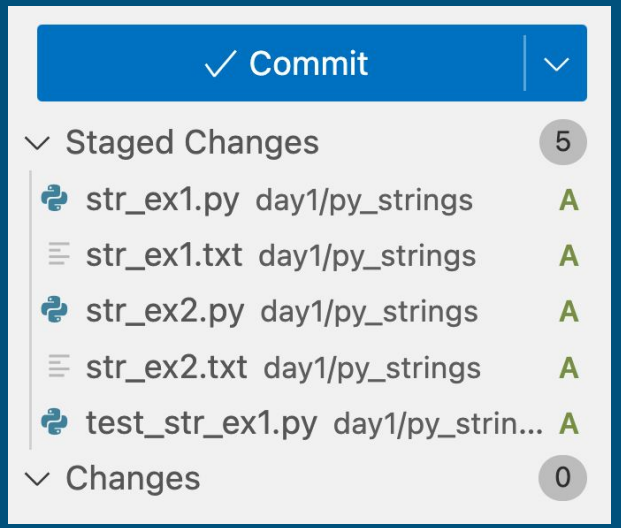
From cmd prompt / terminal, shows that "day1" folder and its contents are not currently being tracked.

Git Add

Adding files (first to staging area, then fully committed).



Stage All Changes.



Files all staged (now)

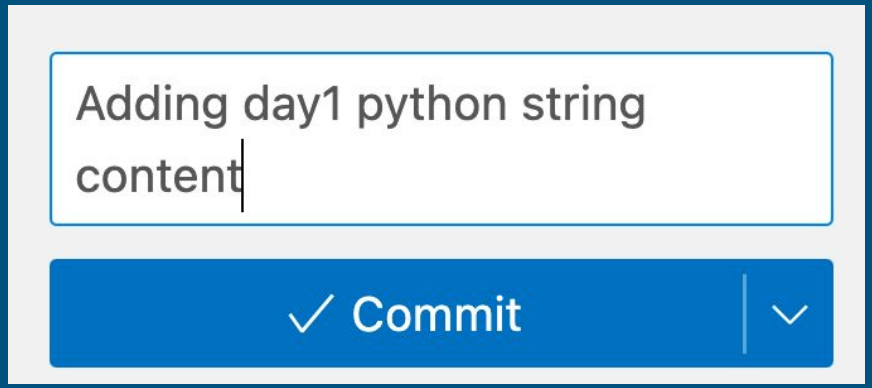
Back to 'git status'

Branch we are working on.

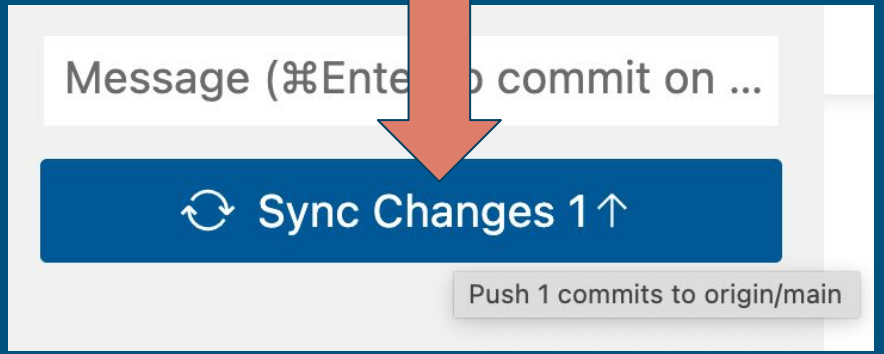
```
(.venv) $ git statu
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   day1/py_strings/str_ex1.py
        new file:   day1/py_strings/str_ex1.txt
        new file:   day1/py_strings/str_ex2.py
        new file:   day1/py_strings/str_ex2.txt
        new file:   day1/py_strings/test_str_ex1.py
```

Process to "unstage" the files.

git commit



'git push' up to GitHub.

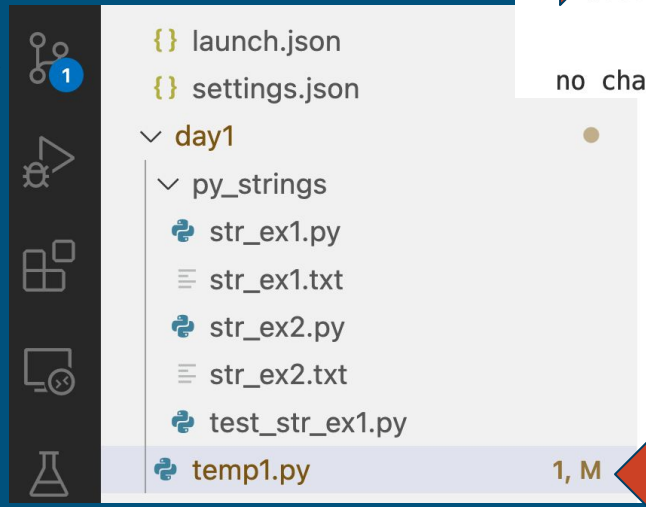


Back to 'git status'

1 file has changed

Process to add or restore file.

```
(.venv) $ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   day1/temp1.py

no changes added to commit (use "git add" and/or "git commit -a")
```

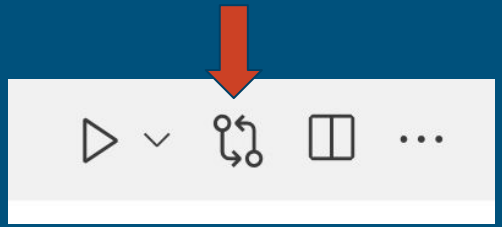


One file modified.

M indicates file modified.

What Changed?

Show what changed in the file.



day1 > temp1.py

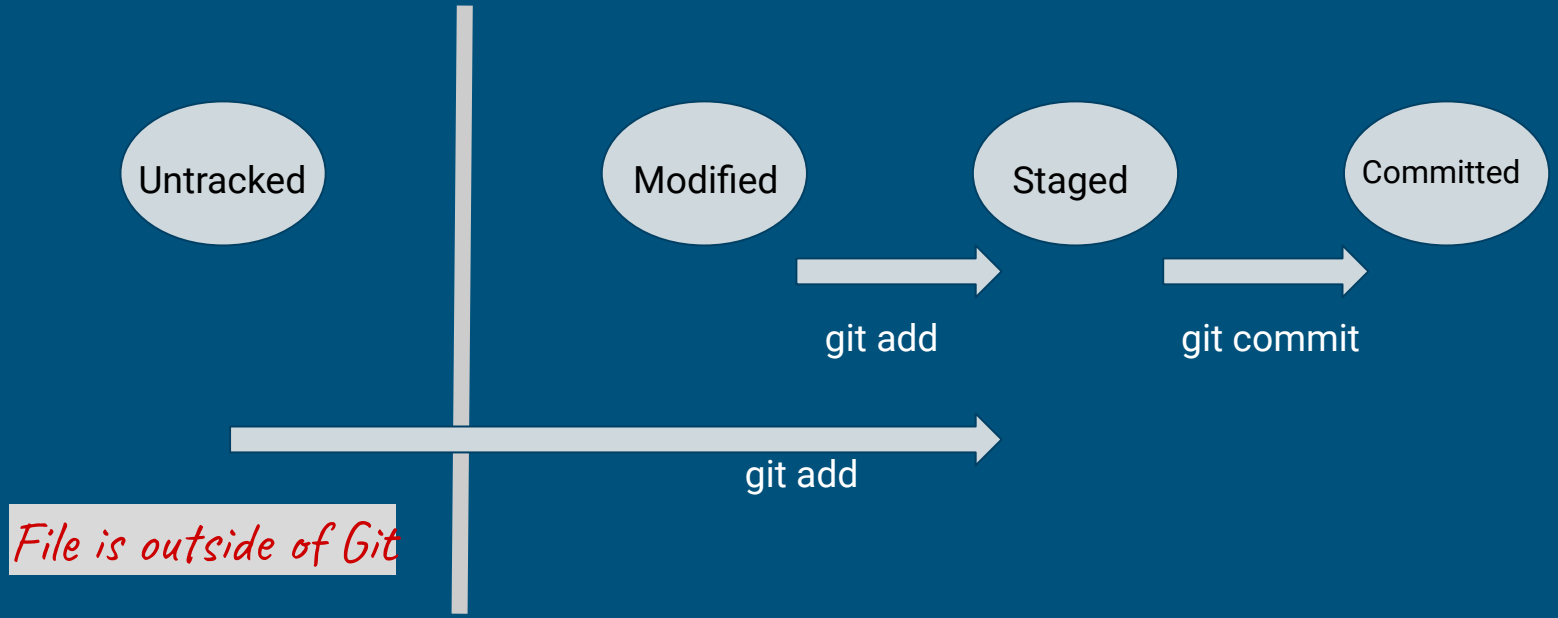
```
1  print("Hello world")
```

```
1   print("Hello world")
2+  print("Something else")
3+  for _ in range(10):
4+      print(_)
5+
```

What Changed?


```
(.venv) $ git diff day1/temp1.py
diff --git a/day1/temp1.py b/day1/temp1.py
index 6d95fe9..dfcd59a 100644
--- a/day1/temp1.py
+++ b/day1/temp1.py
@@ -1 +1,4 @@
-print("Hello world")
\ No newline at end of file
+print("Hello world")
+print("Something else")
+for _ in range(10):
+    print(_)
```

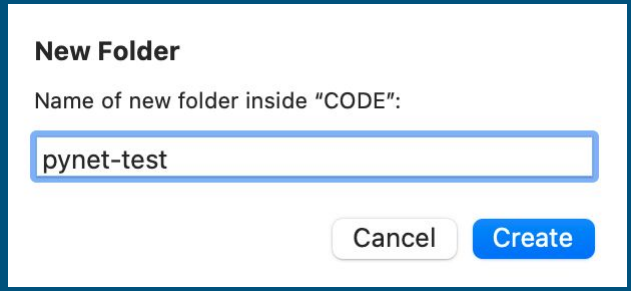
States of your files (git)

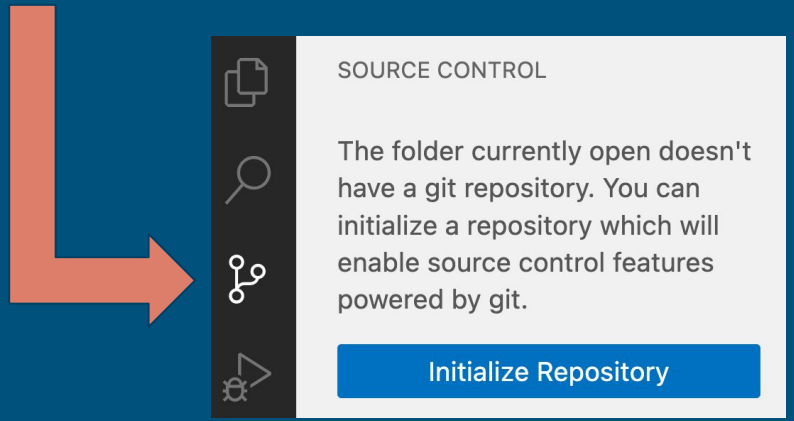


GitHub: {{ repo }}/day1/git/exercise1.txt

Git Exercise1 - Initializing a New Repository

- Close the course's folder in VS Code (file → close folder)
- Select "Start → Open" in VS Code to create a new project. Name this project "pynet-test".
- Click on the "Git" icon in VSCode → Initialize Repository.

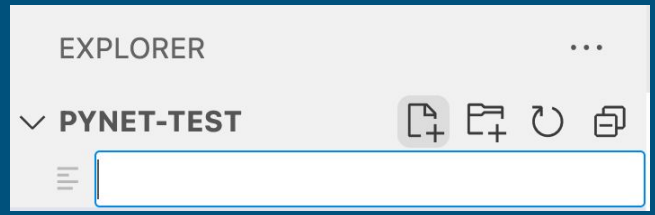


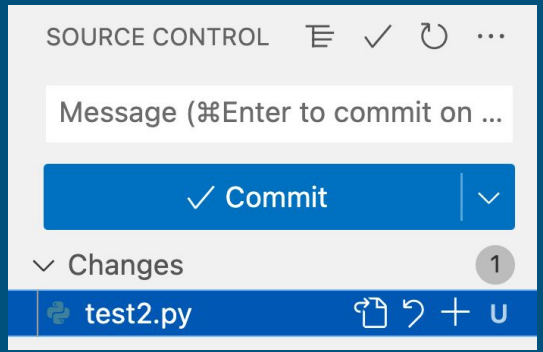


GitHub: {{ repo }}/day1/git/exercise1.txt

Git Exercise1 - Adding a file to repository.

- Check "git status" from the cmd-prompt.
- Using VS Code add a new file to the project.
- Add and commit this file into the Git repository using VS Code (check "git status" from the terminal before commit).

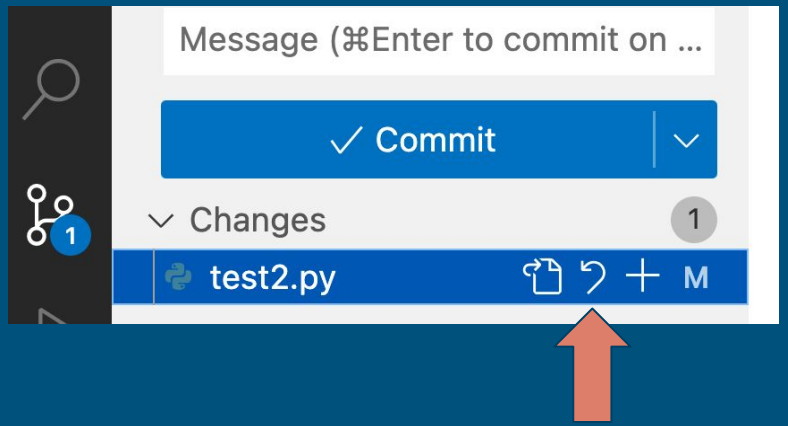




GitHub: {{ repo }}/day1/git/exercise1.txt

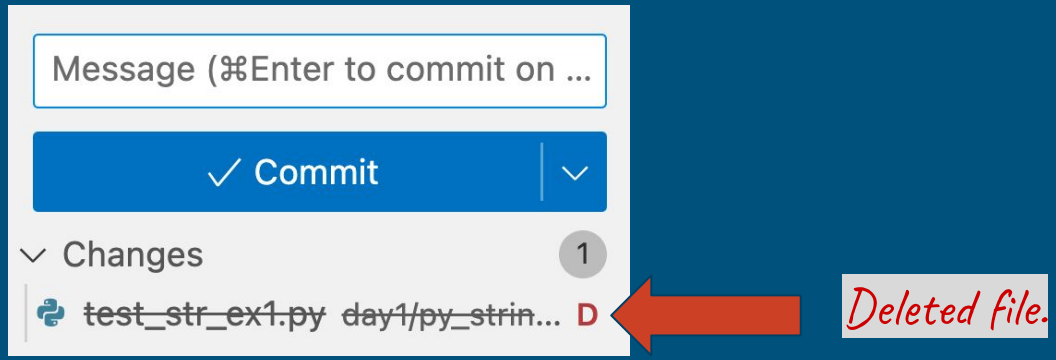
Git Exercise1 - Modifying a file in the repository.

- Modify the file that you previously committed into the repository.
- Using VSCode look at the file's diff (side-by-side comparison).
- Revert this change so the file goes back to its original state.
- Check "git status" that everything is clean after you do this revert.

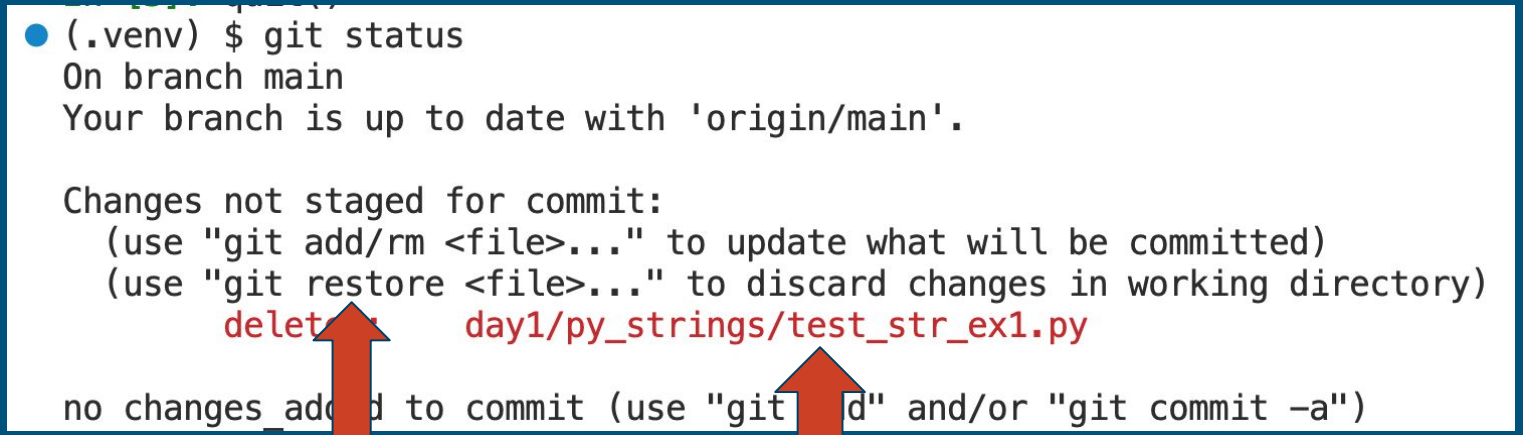


Removing a File

Delete the file in VS Code (right click)



Removing a File



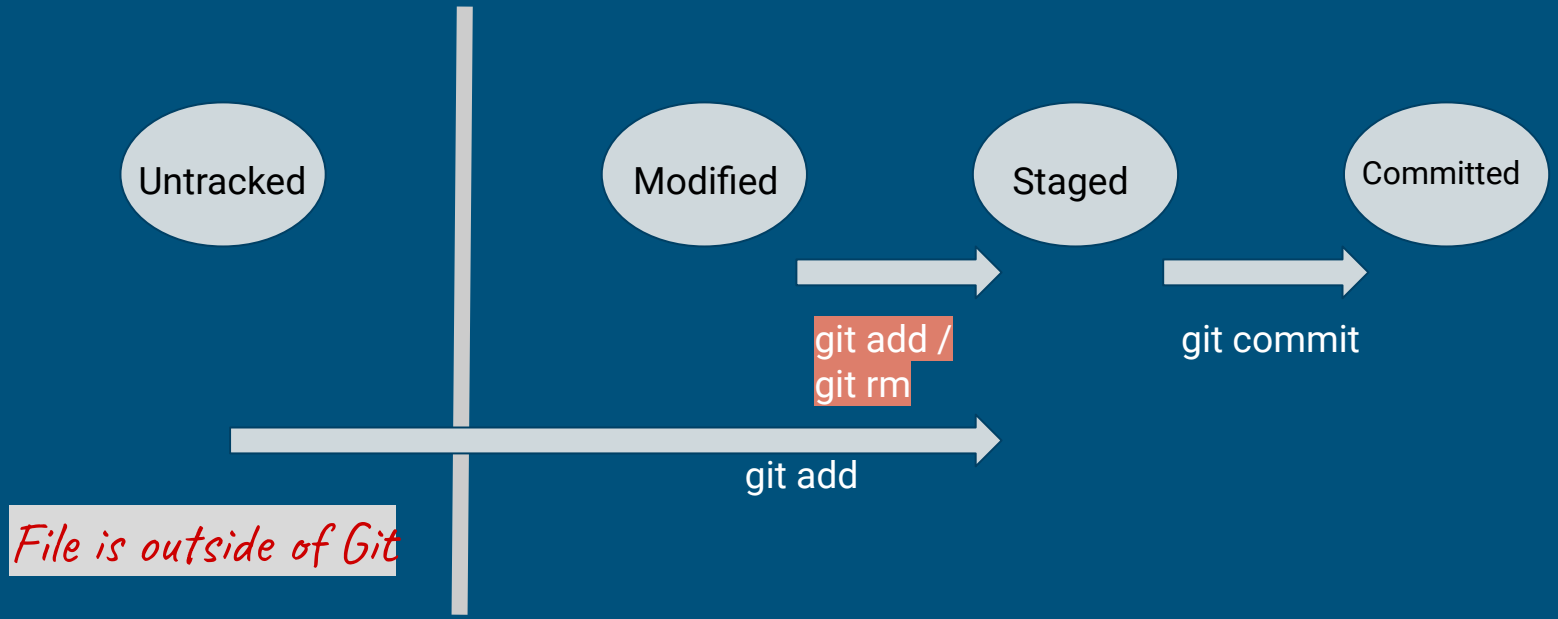
Removing a File


```
(.venv) $ git rm day1/py_strings/test_str_ex1.py
rm 'day1/py_strings/test_str_ex1.py'
(.venv) $ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    day1/py_strings/test_str_ex1.py

(.venv) $ git commit -m "Committing via the command line"
[main 5565149] Committing via the command line
 1 file changed, 14 deletions(-)
 delete mode 100644 day1/py_strings/test_str_ex1.py
```

"git rm" the file (moves to staging state)

Commit the change.

States of your files (git)



What has Happened Across Time


```
commit 556514952ae089e7daee04131f3124b7b988c52c (HEAD -> main, origin/main, origin/HEAD)
Author: Kirk Byers <ktbyers@twb-tech.com>
Date:   Wed Nov 30 09:31:11 2022 -0800

    Committing via the command line

commit 526a6ca58f8902b51ad1ca7dbe8e62cc391fceb9
Author: Kirk Byers <ktbyers@twb-tech.com>
Date:   Wed Nov 30 09:21:02 2022 -0800

    Modifying temp file

commit f544736435f77c2156359162552e2daaf4116e9d
Author: Kirk Byers <ktbyers@twb-tech.com>
Date:   Tue Nov 29 17:16:38 2022 -0800

    Add a temporary file
```

"git log" output

Synchronizing Changes between Systems

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

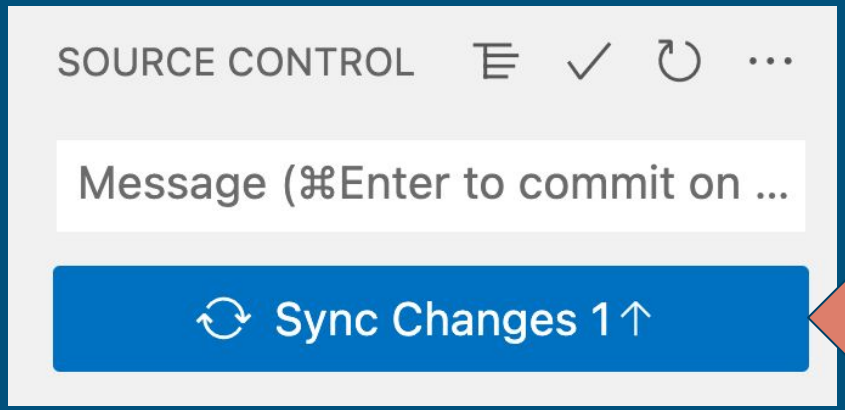
- git pull / git push
- git remote -v
- git remote add
- git branch -vv

"git push" and "git pull"

Reference Commands:

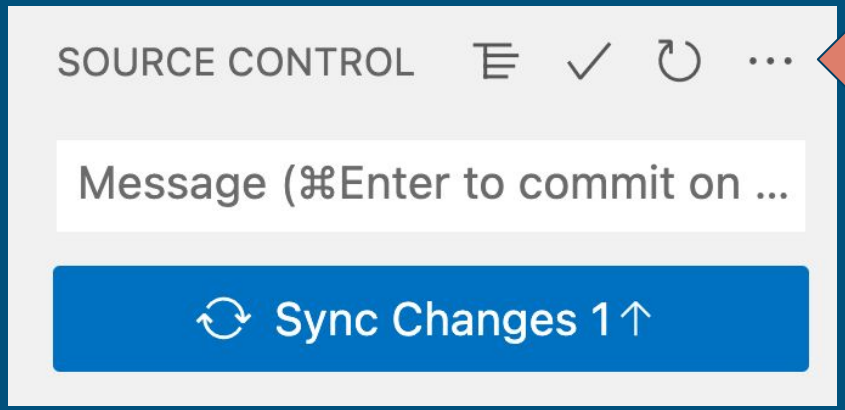
{{ github_repo }}/git_notes/git_commands.MD

Synchronizing Changes between Systems

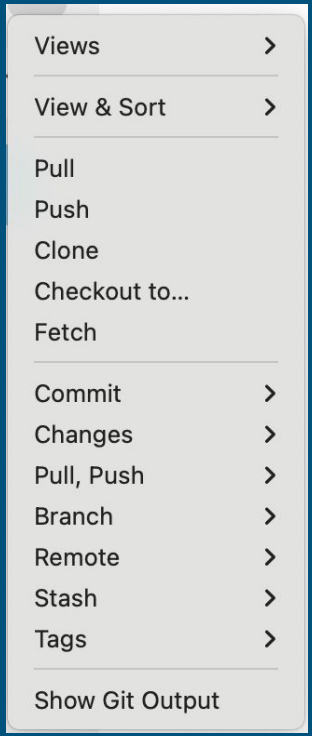


This is a "pull" and a "push".

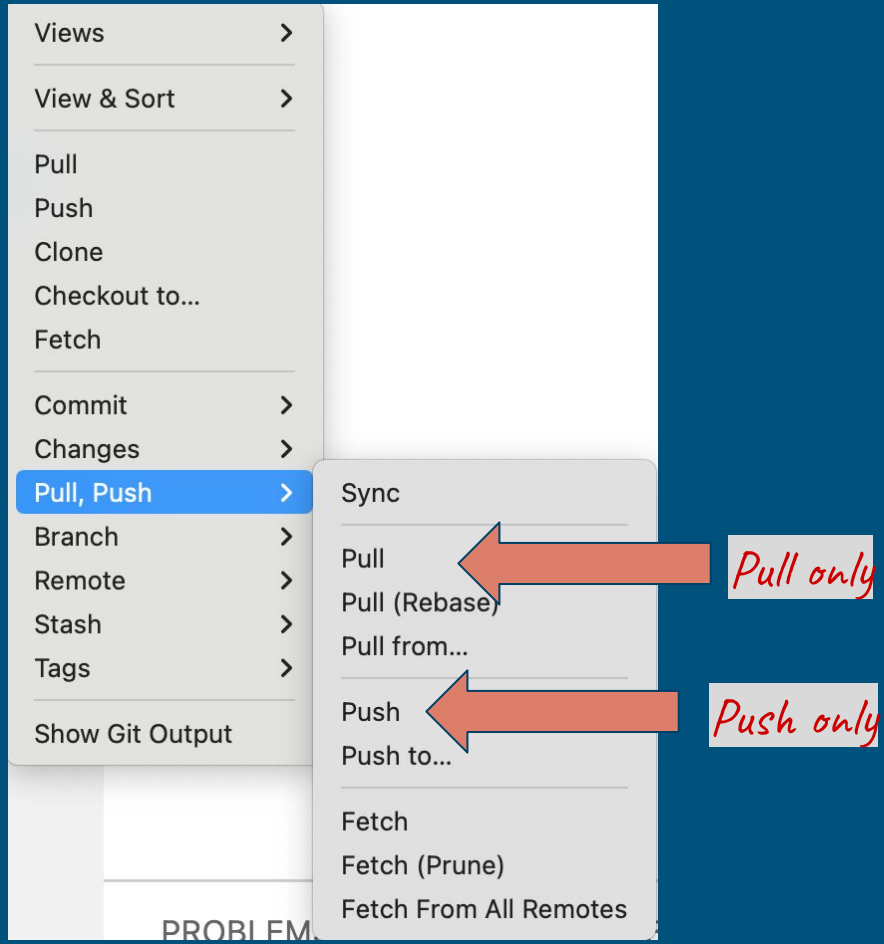
Synchronizing Changes between Systems



Additional "git" options hidden here.



Synchronizing Changes between Systems



Git Remote - Where are my Changes Going?


```
(.venv) $ git remote -v
origin  https://github.com/twin-bridges/pynet-ons-dec22 (fetch)
origin  https://github.com/twin-bridges/pynet-ons-dec22 (push)
```

"origin" is automatically added as a remote on a "git clone"

Adding a Remote

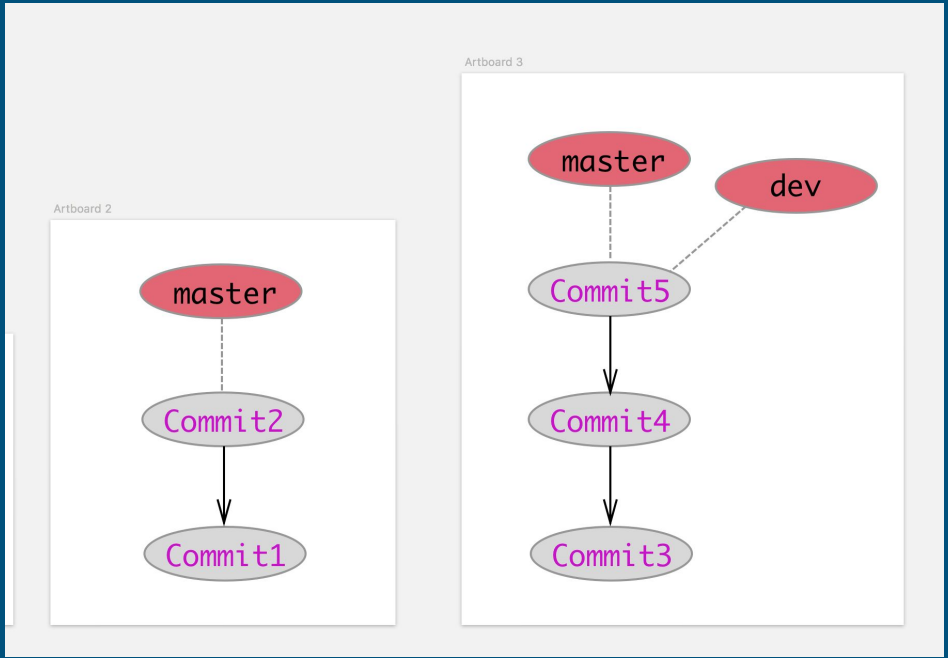
"git remote add"

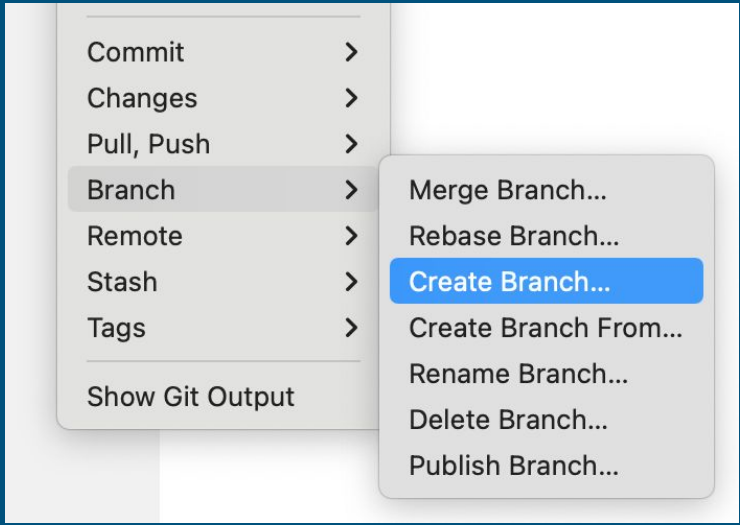
Name of new remote.

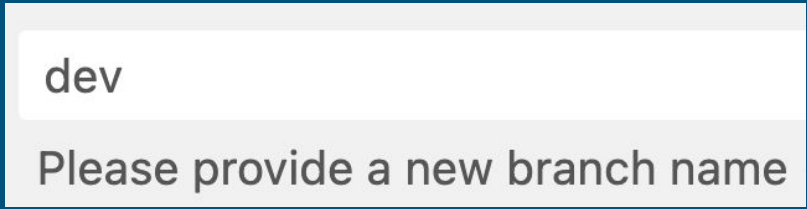
Location of remote


```
(.venv) $ git remote add ktb_remote https://github.com/ktbyers/pynet-ons-dec22
(.venv) $ git remote -v
ktb_remote      https://github.com/ktbyers/pynet-ons-dec22 (fetch)
ktb_remote      https://github.com/ktbyers/pynet-ons-dec22 (push)
origin  https://github.com/twin-bridges/pynet-ons-dec22 (fetch)
origin  https://github.com/twin-bridges/pynet-ons-dec22 (push)
```

Git Branches

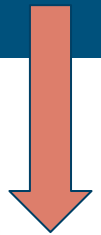






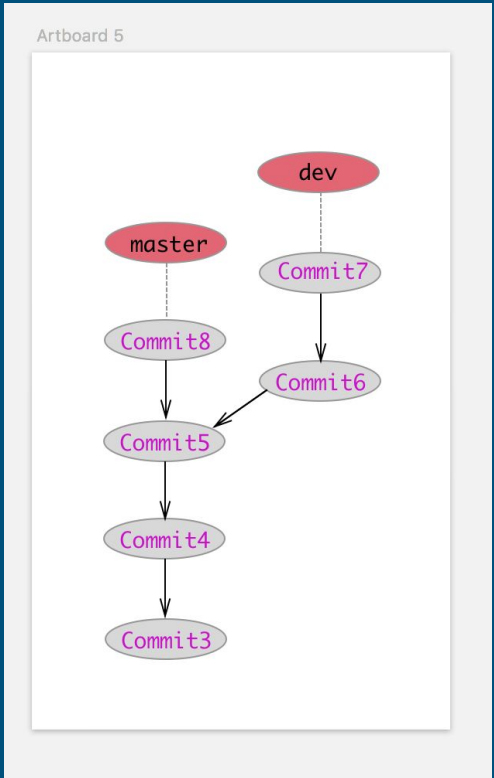
Git Branches

dev, main pointing at the same commit.



```
$ git status
On branch dev
nothing to commit, working tree clean
$ git log
commit aaaec5f7895393e1e65ae02cb92b66c49956f3e5 (HEAD -> dev, main)
Author: Kirk Byers <ktbyers@twb-tech.com>
Date:   Wed Nov 30 11:37:22 2022 -0800

    Adding second test file.
```

Git Branches



After making change to 'dev' branch it is now ahead of the 'main' branch.

```
commit 51f671388242c86e5baa5a1710e759991b5c5d13 (HEAD -> dev)
Author: Kirk Byers <ktbyers@twb-tech.com>
Date:   Wed Nov 30 15:29:58 2022 -0800

    Updating file in dev branch

commit aaaec5f7895393e1e65ae02cb92b66c49956f3e5 (main)
Author: Kirk Byers <ktbyers@twb-tech.com>
Date:   Wed Nov 30 11:37:22 2022 -0800

    Adding second test file.
```

Git Branches

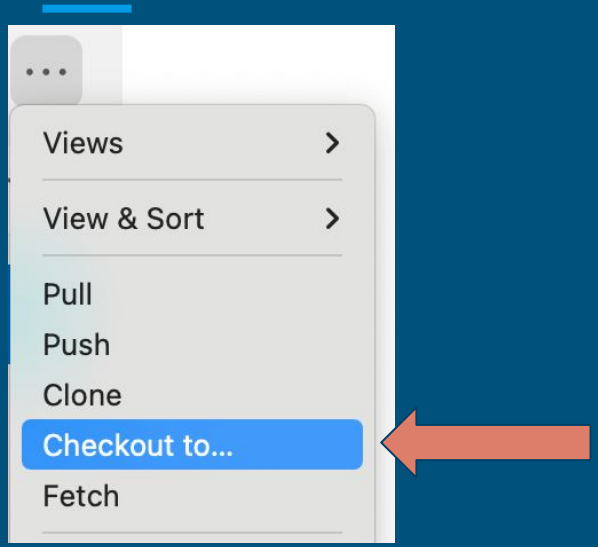
Creating a branch

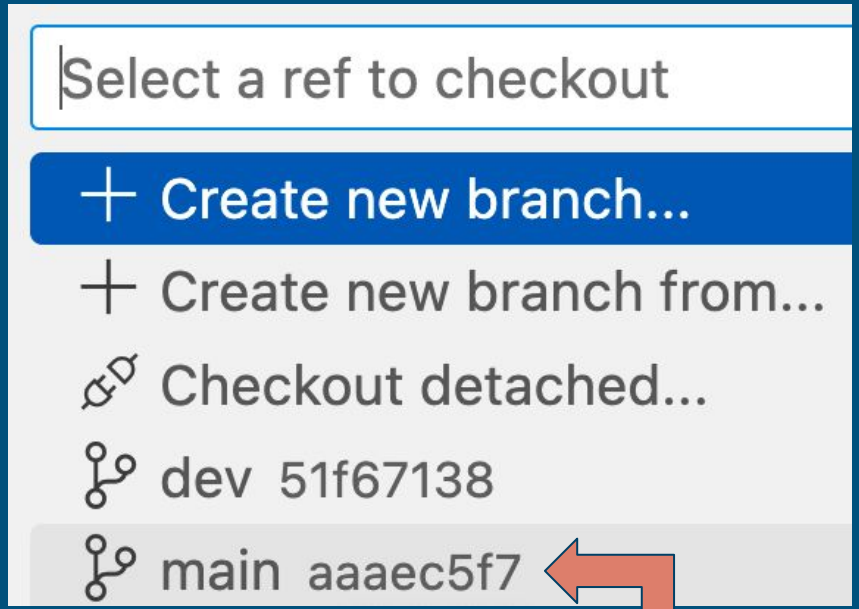
- git checkout -b dev main
- git branch dev2
- git checkout dev2
- git branch *# Look at your current branches*
- Switching branches
    - Underlying files in the working directory change

Merge operation

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

Switching Branches

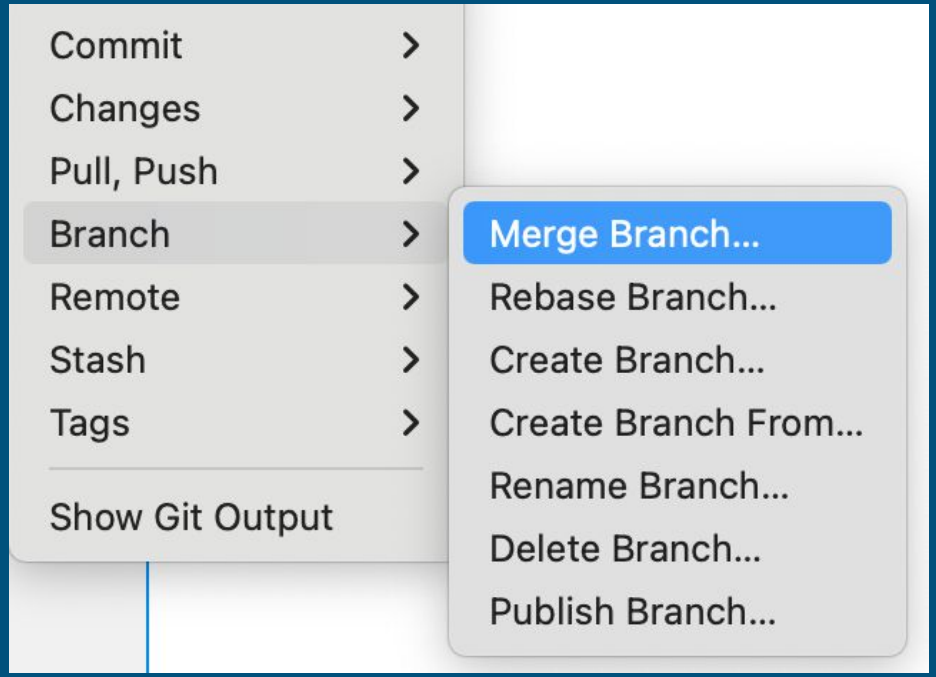




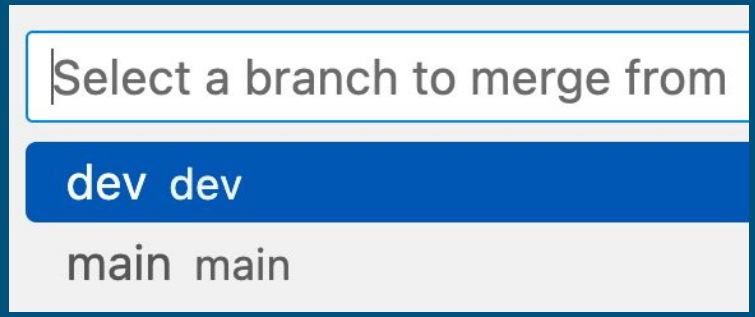
Switch back to the main branch.

Merging Branches

Checkout the branch that you want to merge into (so we will start in "main")



Then pick the branch that you want to merge from.



GitHub: {{ repo }}/day1/git/exercise2.txt

Git Exercise2 - Creating a New Branch

1. Back on your "pynet-test" repository, create a branch named "dev2". Checkout this branch.

2. Using VS Code add a new folder named "day1" containing three files: net1.py, net2.py, and net3.py. What each file contains does not matter.

3. Commit those three files into the "dev2" branch. Using "git log" verify that the "dev2" branch is now at least one commit further than the "main" branch.

4. Merge the "dev2" changes into the "main" branch. Remember you should first checkout the "main" branch and then "merge from" dev2.

5. Using "git log" verify that the "main" branch and the "dev2" branch are now both on the same commit.

6. Using VS Code, verify that your "main" branch now has the three files that you created in the "dev2" branch.

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

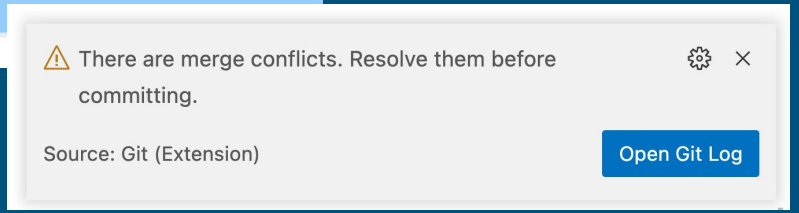
```
while True:
   print("Hello world")
   break

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

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

>>>>>>> dev
```

Git Handling Merge Conflicts

What "main" branch contains.

```
print("Something")
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
<<<<<<< HEAD (Current Change)
for i in range(20):
    print(i)
=======
while True:
    print("zzz")
>>>>>>> dev2 (Incoming Change)
```

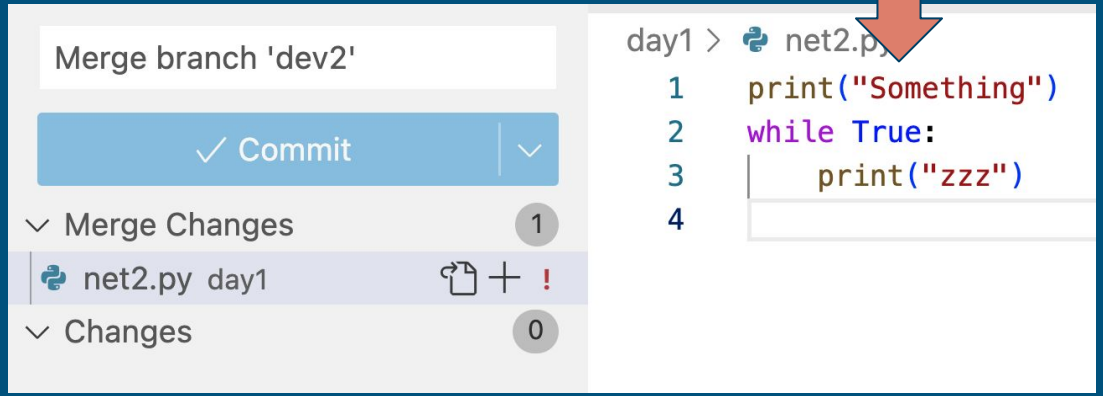
What "dev2" branch contains.



Git Handling Merge Conflicts

Still need to merge that fix in.

Conflict fixed.



Other Git Topics

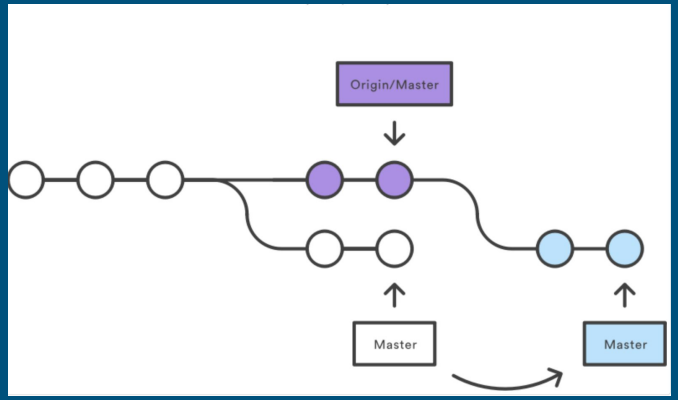
git rebase

git stash

Undoing a committed change.

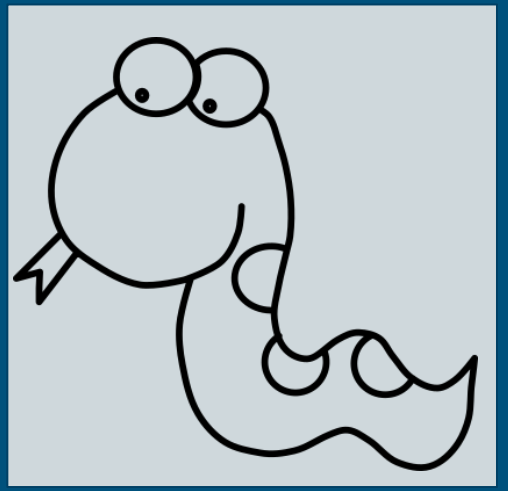
.gitignore file

Pull Requests



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 Characteristics

Indentation matters.

Use spaces not tabs.

Python programmers are particular.

General Items

Should work (hopefully):
Linux, MacOS, Windows

The Python interpreter shell

```
$ python -m IPython
Python 3.11.0 (main, Nov 17 2022, 14:54:18) [GCC 7.3.1 20180712 (Red Hat 7.3.1-15)]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.6.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
```

```
$ python
Python 3.11.0 (main, Nov 17 2022, 14:54:18) [GCC 7.3.1 20180712 (Red Hat 7.3.1-15)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Assignment and variable names

my_var = 22

my_var9 = "some string"

Alphanumeric and underscores

```
In [1]: 9_my_var = "invalid"
  Cell In [1], line 1
    9_my_var = "invalid"
     ^
SyntaxError: invalid decimal literal
```

Variable name cannot lead with a number

sf_ip_address = "10.220.107.99"

By convention, use snake_case for variables

Assignment and variable names


```
In [2]: _ = "my string"
```

Underscore by itself is a valid variable name.

```
In [5]: __name__
Out[5]: '__main__'
```

Variables with leading and trailing double underscores generally have special meaning in Python (dunder-methods or "magic methods")

Variables with a single leading underscore generally means it is a private attribute or private method.

```
In [9]: re._constants
Out[9]: <module 're._constants' from '/usr/local/lib/python3.11/re/_constants.py'>
```

Assignment and variable names


```
In [10]: my_ip_addr = "192.168.88.7"

In [11]: sf_dc_svr9 = "svr9a.lasthop.io"

In [12]: global_delay_factor = 4
```

Python Naming Conventions

snake_case_lower → Used for variable names and functions (can use numbers, just not at the beginning).

PascalCase → Class names.

SNAKE_CASE_UPPER → Used for constants.

self → Reference to object in methods.

_ → Single underscore is a throw-away variable.

Printing to standard output / reading from standard input

```
In [13]: print("Hello world")
Hello world
```

```
In [14]: my_ip_addr = input("Enter an IP Address: ")
Enter an IP Address: 10.220.109.17

In [15]: my_ip_addr
Out[15]: '10.220.109.17'
```

Creating and Executing a Script

Creating a file named "my_code.py"

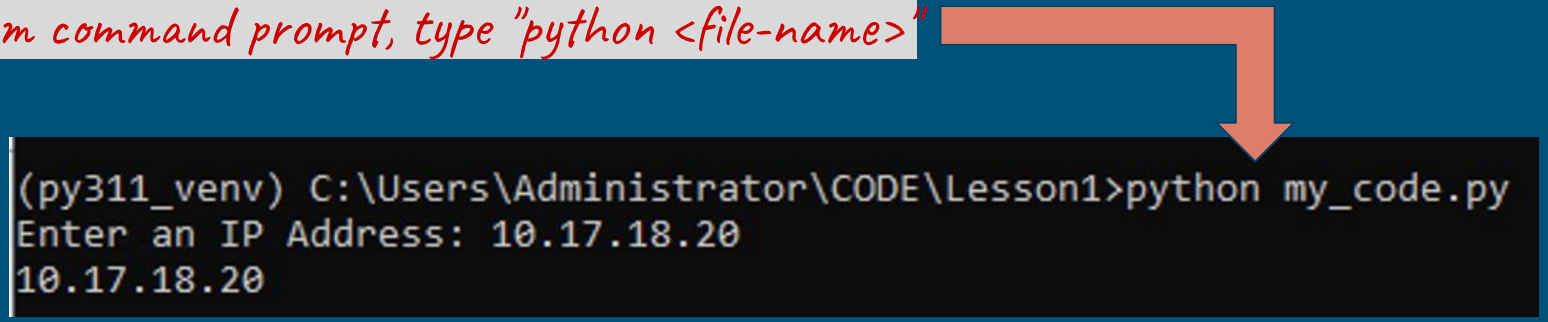
```
my_ip_addr = input("Enter an IP address: ")

print(my_ip_addr)
```

Type "python" and then name of the file.

```
[py3[illegible]_venv] ktbyers@pydev2 ~/learning_python/lesson1
$ python my_code.py
Enter an IP address: 10.17.88.10
10.17.88.10
```

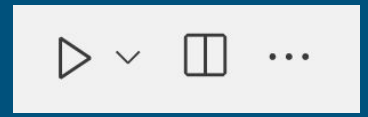
Executing on Windows

From command prompt, type "python <file-name>"



Executing on VS Code

Click "play" button in upper right corner.



Code will execute in the terminal window at the bottom of VS Code.

```
(.venv) $ cd /Users/kirkbyers/CODE/pynet-ons-dec22/day1/scratch
/Users/kirkbyers/CODE/pynet-ons-dec22/.venv/bin/py(.venv) $ /Use
ec22/day1/scratch/my_code.pybyers/CODE/pynet-ons-d
Enter an IP address: 8.8.8.8
8.8.8.8
```

Comments in code


```
# This is a comment
my_ip_addr = input("Enter an IP address: ")

# This line prints out the IP Addr entered above.
print(my_ip_addr)

print("Hello again")  # I can also comment after a statement
```

```
# If you have a comment that spans multiple lines, then just use
# a second comment line.
print(my_ip_addr)
```

Exercise - Creating a Script and Executing It.

GitHub: {{ repo }}/day1/py_general/exercise1.txt

1. In your "pynet-test" repository create a file named my_code.py.
2. This code should prompt a user to enter in a DNS server and save that response to a variable named "dns_server1".
3. You should then print the dns_server1 variable to standard output.
4. Execute this code both from the terminal and from VS Code.
5. Use git to add and commit this file into your repository.

dir() and help()


```
In [4]: my_str = "whatever"

In [5]: dir(my_str)
Out[5]:
['__add__',
 '__class__',
 '__contains__',
```

```
In [6]: help(my_str.upper)
```

```
Help on built-in function upper:

upper() method of builtins.str instance
    Return a copy of the string converted to uppercase.
(END)
```

Python Strings

Can use double quotes.

```
In [1]: my_var = "Some String"

In [2]: my_var2 = 'another string'
```

Or single quotes

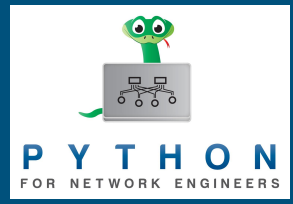
But whatever you start with must be what you end with.

```
In [3]: my_var3 = "Cannot do this'
  Cell In [3], line 1
    my_var3 = "Cannot do this'
                              ^
SyntaxError: unterminated string literal (detected at line 1)
```

PYTHON
FOR NETWORK ENGINEERS

Python Strings

Multiline Strings



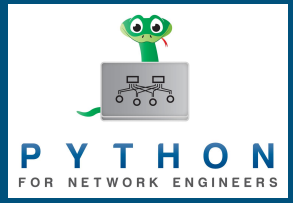

```
In [4]: my_var3 = """Can use triple-quotes
   ...: that
   ...: span
   ...: multiple lines
   ...: """
```

```
In [7]: my_var4 = '''this can be either
   ...: single-quote triple quote or
   ...: double-quote triple quote
   ...: but'''
```

```
In [6]: my_var5 = """You must end
   ...: with what you started with
   ...: ...no can do'''
   ...:
   ...:
   ...: """
```

Doesn't End Until Here

String Methods

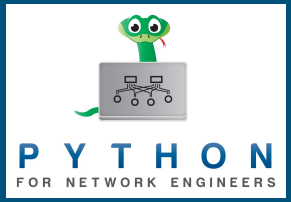



```
In [1]: my_var = "hello"

In [2]: my_var.capitalize()
Out[2]: 'Hello'
```

You can append string methods to the end of strings.

```
In [4]: "something".capitalize()
Out[4]: 'Something'
```

String Methods



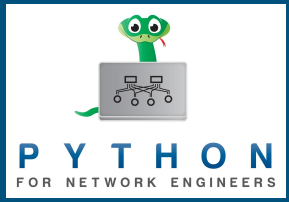
There are a lot of string methods.

```
In [5]: my_var
Out[5]: 'hello'

In [6]: my_var.upper()
Out[6]: 'HELLO'
```

```
In [7]: my_ip = "8.8.8.8"

In [8]: my_ip.split(".")
Out[8]: ['8', '8', '8', '8']
```

```
In [11]: data = "   Some output with leading and trailing whitespace "

In [12]: data.strip()
Out[12]: 'Some output with leading and trailing whitespace'
```

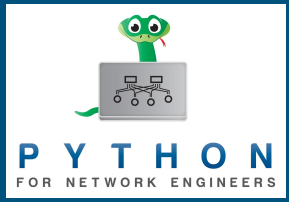
String Methods



In general, string methods do not change the string itself (they return a new string). You would have to do assignment.

```
In [14]: sentence = "this is a sentence."

In [15]: sentence.capitalize()
Out[15]: 'This is a sentence.'

In [16]: sentence
Out[16]: 'this is a sentence.'

In [17]: sentence = sentence.capitalize()

In [18]: sentence
Out[18]: 'This is a sentence.'
```

Useful String Methods



encode()/decode()

format()

strip()/lstrip()/rstrip()

split()

splitlines()

Chaining String Methods

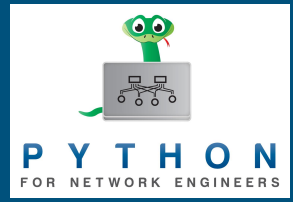
```
In [6]: my_var = "  Some String  "

In [7]: my_var.lower()
Out[7]: '  some string  '

In [8]: my_var.lower().strip()
Out[8]: 'some string'
```

Read left to right

PYTHON
FOR NETWORK ENGINEERS

Exercise

GitHub: {{ repo }}/day1/py_strings/str_ex1.txt



Work in your pynet_test repository

a. Create a python script with a variable called "name". Assign the string "Jane Doe" to this variable.

b. Print the variable to standard output.

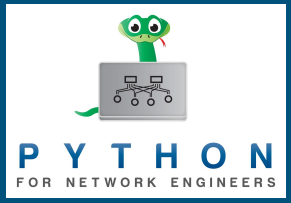
c. Call the .lower() method on this string and print this to standard output.

d. Call the .upper() method on this string and print this to standard output.

e. Call .split() on this string and print this result to standard output.

f. Chain the methods .lower() and then .capitalize() and save this updated string back to the "name" variable. Print this updated name variable to standard out (and verify the variable was in fact changed).

Python f-strings



String interpolation

```
In [4]: print(f"F-strings allow me to embed variables in a string: {ip_addr1}")
F-strings allow me to embed variables in a string: 10.220.89.17
```

```
In [5]: print(f"F-strings technically allow expressions: {2 + 17}")
F-strings technically allow expressions: 19
```

```
In [9]: print(f"F-strings technically allow expressions: {ip_addr1.split('.')[0]}")
F-strings technically allow expressions: 10
```

Python f-strings (creating columns)

PYTHON FOR NETWORK ENGINEERS

Column 20-wide

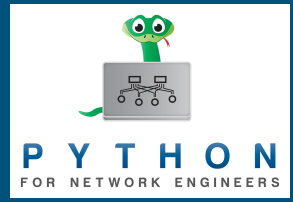
Defaults to left-aligned.

```
In [11]: print(f"{ip_addr1:20}{ip_addr2:20}{ip_addr3:20}")
10.220.89.17         192.168.10.1         172.31.1.9
```

Use > to right align.

```
In [12]: print(f"{ip_addr1:>20}{ip_addr2:>20}{ip_addr3:>20}")
         10.220.89.17         192.168.10.1          172.31.1.9
```

Python f-strings (creating columns)



Column 20-wide

Use ^ to center

```
In [13]: print(f"{ip_addr1:^20}{ip_addr2:^20}{ip_addr3:^20}")
     10.220.89.17         192.168.10.1          172.31.1.9
```

Python f-strings (creating columns)

```
ip_addr1 = "10.220.89.17"
ip_addr2 = "192.168.10.1"
ip_addr3 = "172.31.1.9"
header = "-" * 20
header1 = "ip_addr1"
header2 = "ip_addr2"
header3 = "ip_addr3"

print()
print(f"{header1:^20} {header2:^20} {header3:^20}")
print(f"{header:^20} {header:^20} {header:^20}")
print(f"{ip_addr1:^20} {ip_addr2:^20} {ip_addr3:^20}")
print()
```

```
      ip_addr1            ip_addr2            ip_addr3
-------------------- -------------------- --------------------
    10.220.89.17        192.168.10.1         172.31.1.9
```

Python f-strings (formatting floats)

```
In [15]: my_var = 1/3

In [16]: my_var
Out[16]: 0.3333333333333333

In [17]: f"Value of my_var is: {my_var:.2f}"
Out[17]: 'Value of my_var is: 0.33'
```

f-string Conflicts

```
In [38]: f"Embedded notation can't conflict with string: {my_dict["ip_addr"]}"
  Cell In [38], line 1
    f"Embedded notation can't conflict with string: {my_dict["ip_addr"]}"
                                                          ^
SyntaxError: f-string: unmatched '['
```

You cannot use double-quotes here (as the string was started with double-quotes).

Instead use single-quotes inside the f-string

```
In [39]: f"Embedded notation can't conflict with string: {my_dict['ip_addr']}"
Out[39]: "Embedded notation can't conflict with string: 8.8.8.8"
```

Cases where .format() is still useful

First argument maps to first curly-braces, second to second, third to third.

```
In [14]: "{} {} {}".format(ip_addr1, ip_addr2, ip_addr3)
Out[14]: '10.220.89.17 192.168.10.1 172.31.1.9'
```

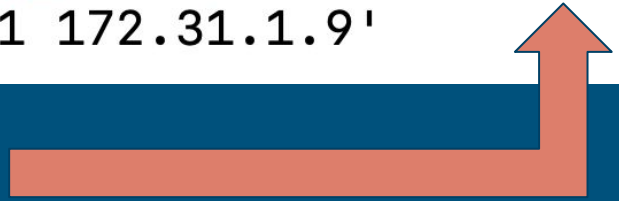
```
In [18]: my_list
Out[18]: ['10.220.89.17', '192.168.10.1', '172.31.1.9']

In [19]: "{} {} {}".format(*my_list)
Out[19]: '10.220.89.17 192.168.10.1 172.31.1.9'
```

Unpacking

Cases where .format() is still useful

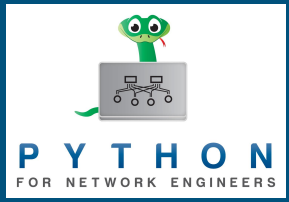
```
In [20]: my_dict
Out[20]:
{'ip_addr1': '10.220.89.17',
 'ip_addr2': '192.168.10.1',
 'ip_addr3': '172.31.1.9'}

In [21]: "{ip_addr1} {ip_addr2} {ip_addr3}".format(**my_dict)
Out[21]: '10.220.89.17 192.168.10.1 172.31.1.9'
```

Unpacking, but with key-value pairs.



Exercise

GitHub: {{ repo }}/day1/py_strings/str_ex2.txt



Work in your pynet_test repository

a. Create a python script with three strings representing three names.

b. Print these three names out in a column 30 wide, right aligned.

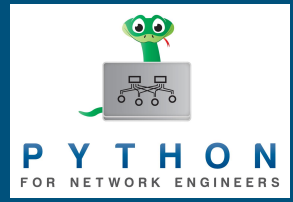
c. Execute the script verify the output.

d. Add a prompt for a fourth name, print this out as well.

e. Check your code into Git

Exercise

GitHub: {{ repo }}/day1/py_strings/str_ex3.txt



Work in your pynet_test repository

a. Create a python script that prompts for an IP address.

b. split on '.'

c. Print out four octets with column width of 12; left aligned.

d. Check your code into git

Strings are "sequences"

What do I mean by sequence:

1. They have an order: first letter, second letter, etc.
2. You can access the individual letters by using an index.
3. Strings have a length.
4. You can loop over a string and obtain the letters.

Strings are "sequences"

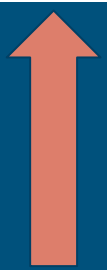

```
In [10]: my_var
Out[10]: 'some string'
```

```
In [11]: my_var[0]
Out[11]: 's'

In [12]: my_var[1]
Out[12]: 'o'
```

```
In [14]: for letter in my_var:
    ...:     print(letter)
    ...:
s
o
m
e

s
t
r
i
n
g
```

Strings Membership

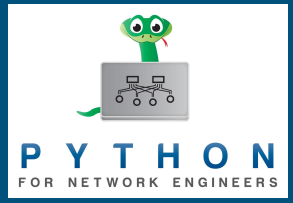
```
In [1]: some_str = "It was the best of times, it was the worst of times"

In [2]: "the best" in some_str
Out[2]: True
```



Checks if substring is in larger string.

Returns a Boolean: either True or False

Raw Strings




```
In [3]: win_path = "c:\windows\new_dir\test\applications"

In [4]: print(win_path)
c:\windows
ew_dir  estpplications
```

Some characters like \n and \t have special meaning and hence would need either escaped or use a raw string.

```
In [5]: win_path = r"c:\windows\new_dir\test\applications"

In [6]: print(win_path)
c:\windows\new_dir\test\applications
```

Numbers

```
In [1]: my_var = 22

In [2]: type(my_var)
Out[2]: int
```

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

```
In [3]: my_var = 22.7

In [4]: type(my_var)
Out[4]: float
```

```
In [6]: 17 + 22
Out[6]: 39

In [7]: 22 - 7
Out[7]: 15

In [8]: 3 * 4
Out[8]: 12

In [9]: 4 / 7
Out[9]: 0.5714285714285714
```

Exercises:
./day1/py_numbers/numbers_ex1.txt

Lists

Similar to an array in PowerShell

Creating a List

Square bracket notation.

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

Separate the elements using commas.

```
In [2]: type(my_list)
Out[2]: list
```

Data types for the elements of the list can vary (strings, integers, booleans, other lists, etc)

Lists

Lists use zero-based indices

```
In [4]: my_list
Out[4]: ['foo', 1, 'hello', [], None, 2.7]

In [5]: my_list[0]
Out[5]: 'foo'
```

Accessing the first element (element zero)

```
In [98]: my_list[0] = 88
```

Can assign new values.

Lists are sequential...first element, second element, third element, etc.

Accessing the last element.

```
In [7]: my_list
Out[7]: ['foo', 1, 'hello', [], None, 2.7]

In [8]: my_list[-1]
Out[8]: 2.7
```

Accessing the last element (using a -1 index)

You can use -1, -2, -3, to work backwards from the end of the list.

Length and Range

Length of the list.

```
In [84]: my_list
Out[84]: [1, 'hello', [22], None, 2.7, 'new string', 'zzz']

In [85]: len(my_list)
Out[85]: 7
```

```
In [87]: list(range(10))
Out[87]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
In [90]: list(range(1, 5))
Out[90]: [1, 2, 3, 4]
```

You can change the starting value.

Range generates a sequence of numbers from 0 to (n-1). Note, this exactly corresponds to the indices of list of length n.

List Membership

```
In [2]: my_list
Out[2]: ['Some', 'set', 'of', 'values', 22, -4.7, None]

In [3]: "values" in my_list
Out[3]: True

In [4]: "whatever" in my_list
Out[4]: False
```

Does the given element exist in the list or not (returns a boolean).

List Methods

append()
clear()
copy()
count()
extend()
index()
insert()
pop()
remove()
reverse()
sort()

```
In [13]: my_list
Out[13]: ['foo', 1, 'hello', [], None, 2.7]

In [14]: my_list.append("new string")

In [15]: my_list
Out[15]: ['foo', 1, 'hello', [], None, 2.7, 'new string']
```

Modifies the existing list–why?

List Methods

append()
clear()
copy()
count()
extend()
index()
insert()
pop()
remove()
reverse()
sort()

```
In [20]: my_list
Out[20]: ['foo', 1, 'hello', [], None, 2.7, 'new string']

In [21]: my_list.clear()

In [22]: my_list
Out[22]: []
```

```
In [33]: my_list
Out[33]: ['foo', 1, 'hello', [], None, 2.7, 'new string']

In [34]: my_list.count("hello")
Out[34]: 1
```

List Methods

append()
clear()
copy()
count()
extend()
index()
insert()
pop()
remove()
reverse()
sort()

This is a shallow copy.

```
In [27]: my_list
Out[27]: ['foo', 1, 'hello', [], None, 2.7, 'new string']

In [28]: new_list = my_list.copy()

In [29]: new_list
Out[29]: ['foo', 1, 'hello', [], None, 2.7, 'new string']

In [30]: id(my_list)
Out[30]: 4413908288

In [31]: id(new_list)
Out[31]: 4414123072
```

Not the same thing. Once again relates to lists being mutable.

python™

List Methods - Concatenation

append()
clear()
copy()
count()
extend()
index()
insert()
pop()
remove()
reverse()
sort()

We can concatenate strings.

```
In [41]: "hello" + " world"
Out[41]: 'hello world'

In [42]: my_list
Out[42]: ['foo', 1, 'hello', [22], None, 2.7, 'new string']

In [43]: my_list + [22, "whatever"]
Out[43]: ['foo', 1, 'hello', [22], None, 2.7, 'new string', 22, 'whatever']
```

We can also concatenate lists.

List Methods - Concatenation

append()
clear()
copy()
count()
extend()
index()
insert()
pop()
remove()
reverse()
sort()

```
In [45]: my_list.extend(['zzz', 42])

In [46]: my_list
Out[46]: ['foo', 1, 'hello', [22], None, 2.7, 'new string', 'zzz', 42]
```

Essentially concatenating lists here.

Pop things from a list.

append()
clear()
copy()
count()
extend()
index()
insert()
pop()
remove()
reverse()
sort()

```
In [49]: my_list
Out[49]: ['foo', 1, 'hello', [22], None, 2.7, 'new string', 'zzz', 42]

In [50]: val1 = my_list.pop()

In [51]: my_list
Out[51]: ['foo', 1, 'hello', [22], None, 2.7, 'new string', 'zzz']

In [52]: val1
Out[52]: 42
```

Remove and return the last value from the list.

Pop things from a list.

Remove and return the very first value.

append()
clear()
copy()
count()
extend()
index()
insert()
pop()
remove()
reverse()
sort()

```
In [53]: my_list
Out[53]: ['foo', 1, 'hello', [22], None, 2.7, 'new string', 'zzz']

In [54]: val2 = my_list.pop(0)

In [55]: my_list
Out[55]: [1, 'hello', [22], None, 2.7, 'new string', 'zzz']

In [56]: val2
Out[56]: 'foo'
```

List Exercise1

Exercises:
./day1/py_lists/lists_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

f. Access index-0 (my_list[0]) and assign it a new value.

List Exercise2

Exercises:
./day1/py_lists/lists_ex2.txt

a. Create two lists named list1 and list2 containing various strings, integers, and floats.

b. Create a list3 by concatenating list1 and list2 together.

c. Use rich.print to print out this list3.

d. Instead of using list concatenation now directly modify list1 by using list1.extend(list2)

e. Pop off the very first element of list3 and save it to a variable named first_element. Print out this variable. Also verify list3 has changed (i.e. no longer has the first element).

Join - Not a list method, but uses lists.

.split() takes a string and from that creates a list based on a separator character(s)

```
In [62]: ip_addr = "192.168.10.1"

In [63]: ip_addr.split(".")
Out[63]: ['192', '168', '10', '1']
```

.join() takes a separator string and feeds in a list. Puts separator between each list entry.

```
In [65]: fields
Out[65]: ['192', '168', '10', '1']

In [66]: ".".join(fields)
Out[66]: '192.168.10.1'
```

List Slices - Enough with the lists already.

List slices is a way to create new lists from parts of an existing list.

```
In [67]: my_list
Out[67]: [1, 'hello', [22], None, 2.7, 'new string', 'zzz']

In [68]: my_list[1:3]
Out[68]: ['hello', [22]]
```

List slice.

First index is included.

Second index is excluded.

List Slices - Dynamically create new lists.

No first index = start at the beginning of the list.

```
In [69]: my_list[:3]
Out[69]: [1, 'hello', [22]]
```

```
In [71]: my_list[4:]
Out[71]: [2.7, 'new string', 'zzz']
```

No last index = go to the end of the list.

Another way to copy a list.

```
In [72]: my_list[:]
Out[72]: [1, 'hello', [22], None, 2.7, 'new string', 'zzz']
```

List Slices - Can use negative indices.

Remember second number of list slice is excluded (so here the last entry in list is dropped from the new list).

```
In [81]: my_list[4:-1]
Out[81]: [2.7, 'new string']
```

Tuple - An Immutable List

Parenthesis, not brackets

```
In [82]: my_tuple = (1, "hello", 22, None, 2.7)

In [83]: type(my_tuple)
Out[83]: tuple
```

```
In [92]: my_tuple[2]
Out[92]: 22
```

Still access using indices using [] notation.

Tuple - An Immutable List

Cannot assign new values.

```
In [102]: my_tuple[2] = 44
---------------------------------------------------------------
TypeError                                 Traceback (most
Cell In[102], line 1
----> 1 my_tuple[2] = 44

TypeError: 'tuple' object does not support item assignment
```

Cannot append(), extend(), pop()

```
In [103]: my_tuple.append(88)
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[103], line 1
----> 1 my_tuple.append(88)

AttributeError: 'tuple' object has no attribute 'append'
```

List Exercise3

Exercises:
./day1/py_lists/lists_ex3.txt

1. In your Python program, create the following string.

sentence = "This is a sentence consisting of eight words."

2. Use the split() method to create a new variable containing a list of the words in the sentence.

3. Change the first element of your newly created list to be all lower case.

4. Change the last element of your newly created list to be just "words" i.e. no period at the end.

5. Check whether the word "eight" exists in the list.

List Exercise4

Exercises:
./day1/py_lists/lists_ex4.txt

1. Use list(range(1,50)) to create a new list named my_list.

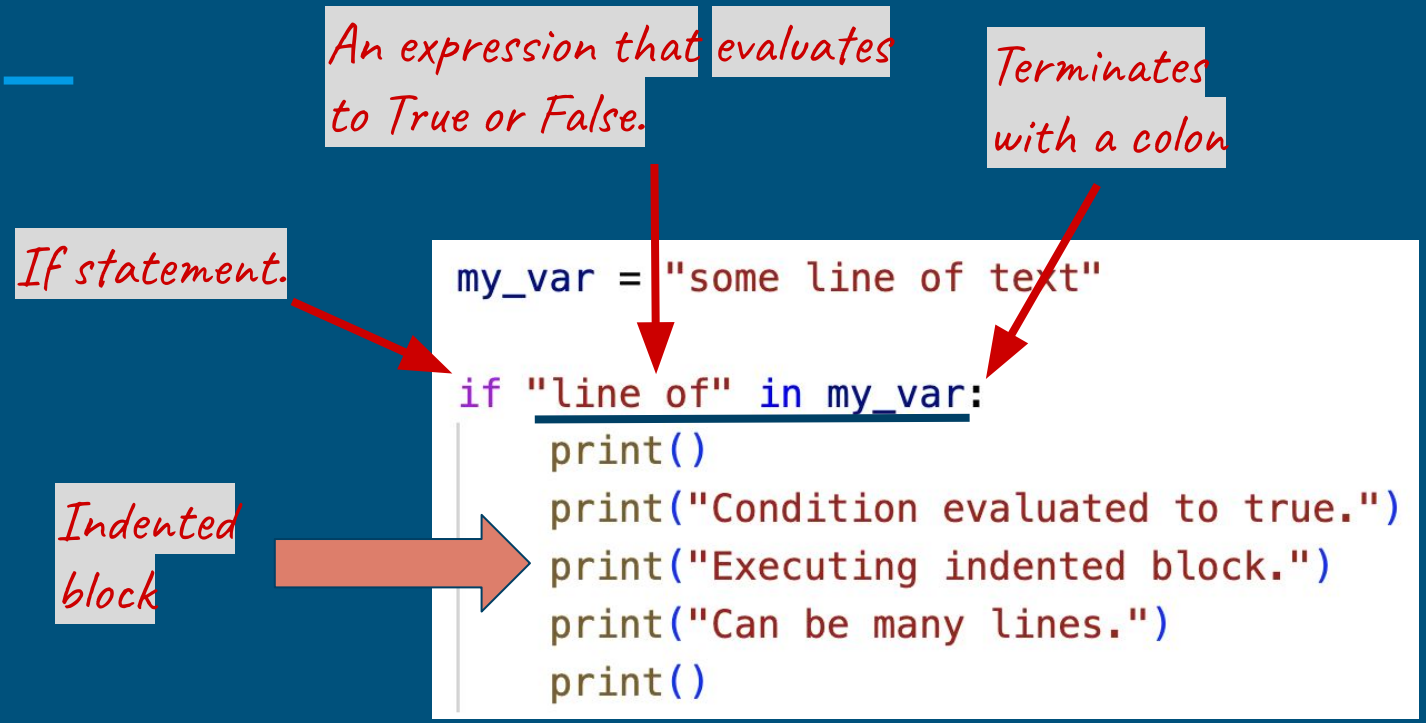
2. Add "hello" and "world" onto the end of the list (as two new elements).

3. Change index-0 to be "whatever". Use rich.print() to print out your current list.

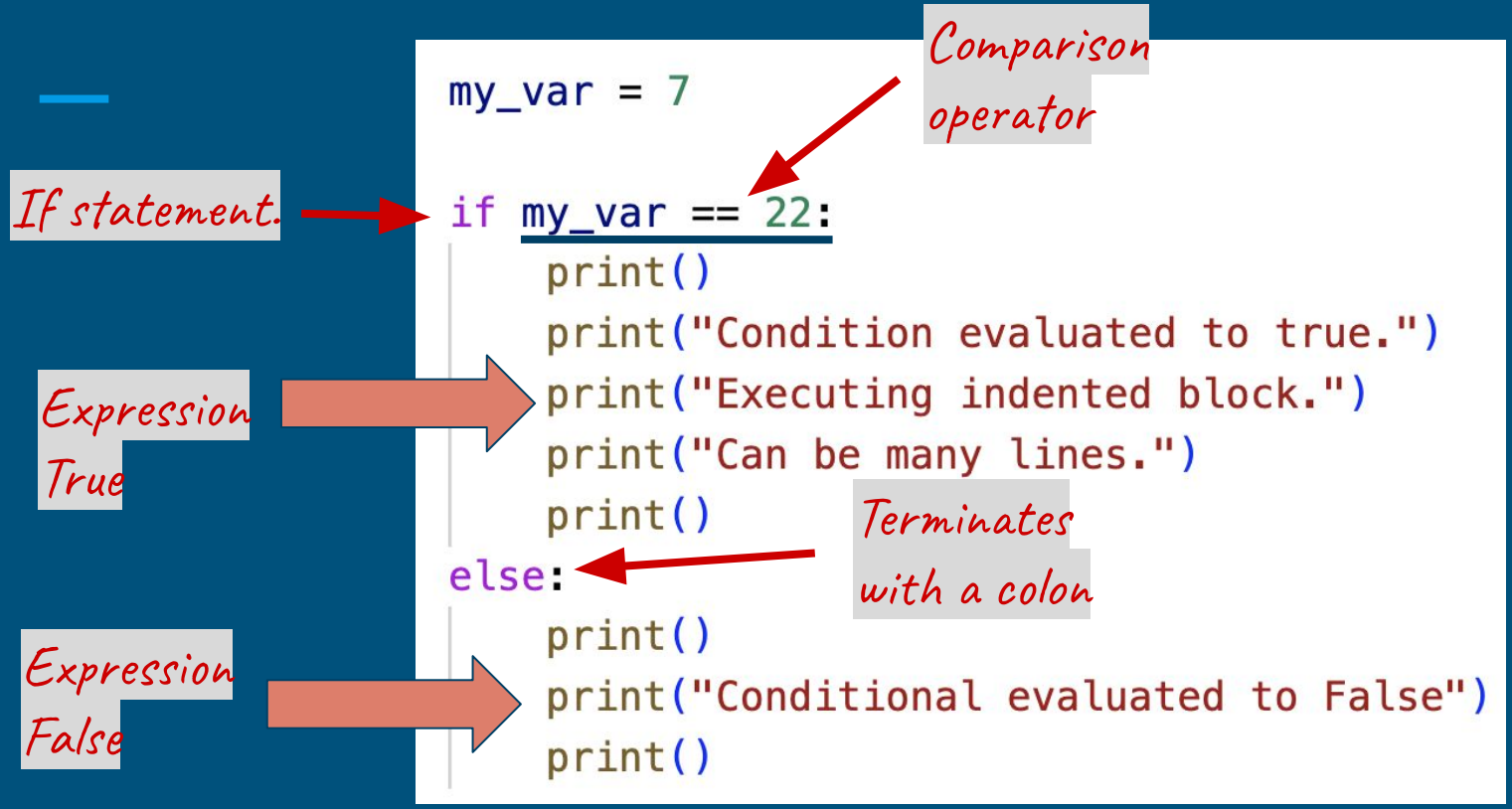
4. Create a new list using a list slice where the new list is the last three elements of my_list. Use rich.print() to print out this new list.

5. Make a second new list using a list slice where this list is the first seven elements of my_list. Print out this new list.

Making Choices - Conditionals



Making Choices - Conditionals




```
my_var = 7

if my_var == 22:
    print()
    print("Condition evaluated to true.")
    print("Executing indented block.")
    print("Can be many lines.")
    print()
else:
    print()
    print("Conditional evaluated to False")
    print()
```

Making Choices - Chaining "ifs"

First True expression will be the one that executes (otherwise the "else" section will execute).

Only one indented block will executed.

```
my_list = [22, 42, "hello", "world", "whatever"]

if "something" in my_list:
    print()
    print("Found something")
    print()
elif "hello" in my_list:
    print()
    print("Found hello")
    print()
elif "nothing" in my_list:
    print()
    print("Found nothing")
    print()
else:
    print("Strings not detected")
```

Comparison Operators

==	Equal
!=	Not Equal
>	Greater Than
>=	Greater Than or Equal
<	Less Than
<=	Less Than or Equal

```
In [3]: a = 2

In [4]: b = 7

In [5]: a == b
Out[5]: False
```

```
In [8]: a
Out[8]: 2

In [9]: b
Out[9]: 7

In [10]: a != b
Out[10]: True
```

```
In [14]: a < b
Out[14]: True

In [15]: a > b
Out[15]: False
```

```
In [16]: a = "hello"

In [17]: b = "hello"

In [18]: a == b
Out[18]: True
```

Logical and

Both expressions must be True for the if-statement to execute.

```
my_var = 22
if my_var >= 10 and my_var <= 100:
    print("\nValue is between 10 and 100\n")
```

Logical or

Either expression can be True (and then if-statement will execute).

```
my_var = 42
if my_var == 10 or my_var == 42:
    print("\nmy_var is either 10 or 42\n")
```

"Not" Expression

not (expression) flips expression from True to False and vice versa.

```
In [1]: a = True

In [2]: not a
Out[2]: False
```

```
my_var = False
if not my_var:
    print("\nmy_var is False\n")
```

Conditional Exercise1

Exercises:
./day1/py_conditionals/ex1.txt

1. Prompt to enter an IP address. Save this to a variable named ip_addr.

2. Check if "192.168" is a part of the IP address (substring in broader string). Don't worry about whether at the beginning, middle, or end of the IP address.

3. If "192.168" is a part of the IP address, then print "The IP Address is Correct".

4. If "192.168" is not a part of the IP address, then print out "Invalid IP Address" and include the incorrect IP address as part of the message.

Conditional Exercise2

Exercises:
./day1/py_conditionals/ex2.txt

1. Create a variable named os_version with the following value:

os_version = "15.4(2)T1"

2. Using the .split() method, extract the major version 15 and the minor version 4. Note, you potentially will have to use split()twice to extract the minor version.

3. Check if the major version is version 16 or greater. Print a message if this is True.

4. If that fails (i.e. major version is less than 16), then check if the major version is 15, and the minor version is 2 or greater (i.e. 15.2 or greater). Print a message if this is True.

5. If neither of these conditions are true, print a message indicating that the os_version is invalid.

/* Note, you will likely need to cast the string as integers using int(my_var) */

8.

```
In [1]: my_var = "15"

In [2]: int(my_var)
Out[2]: 15
```

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