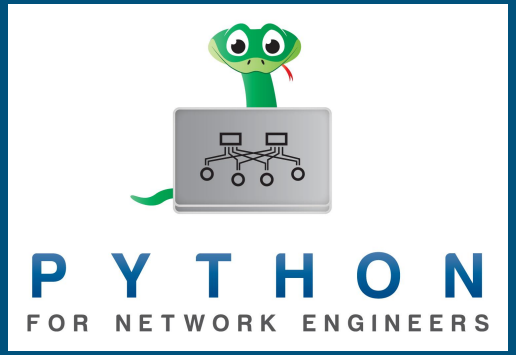
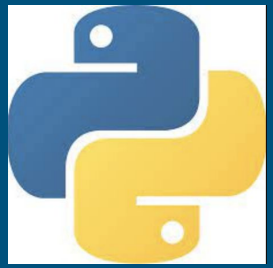




Python Automation Training Session
Dec 2022

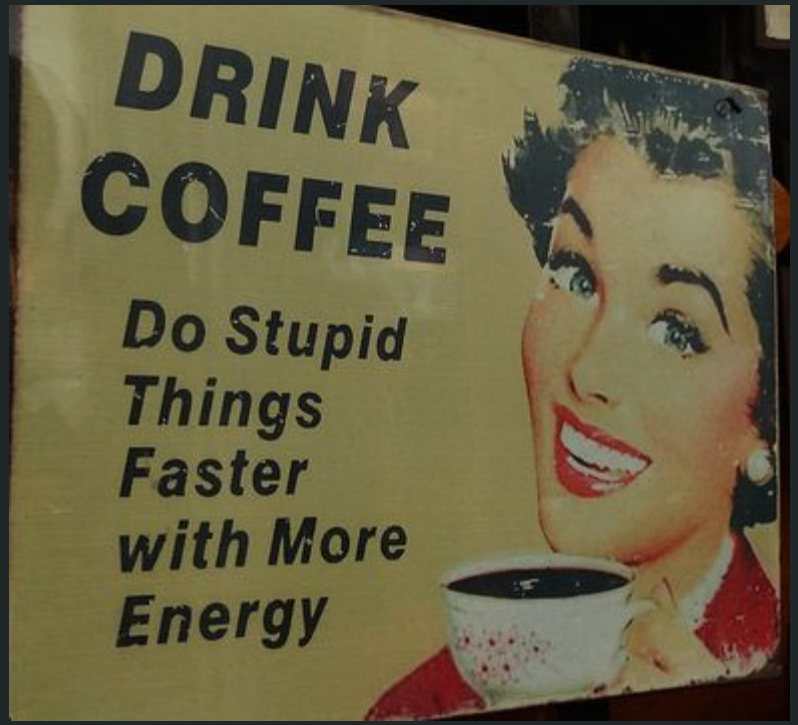
https://github.com/twin-bridges/pynet-ons-dec22/blob/main/python-training-day2.pdf

General:

Virtual-Training Day
Dec 12th, Day2 (Mon) / 9AM - 4:30PM Central

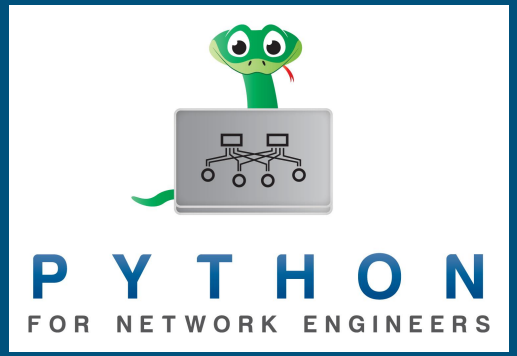
Focused/Minimize Distractions

The exercises are important.



Flickr: Ben Sutherland

Day2 Training Schedule



Python Review

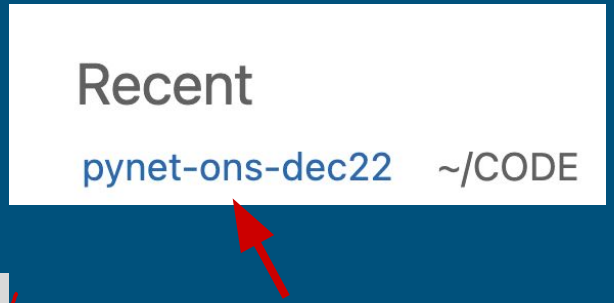
Python Fundamentals:

* Lists
* Booleans
* Conditionals
* Files
* Loops
* Dictionaries
* Exceptions

VS Code Debugging

Environment Check

VS Code → Open

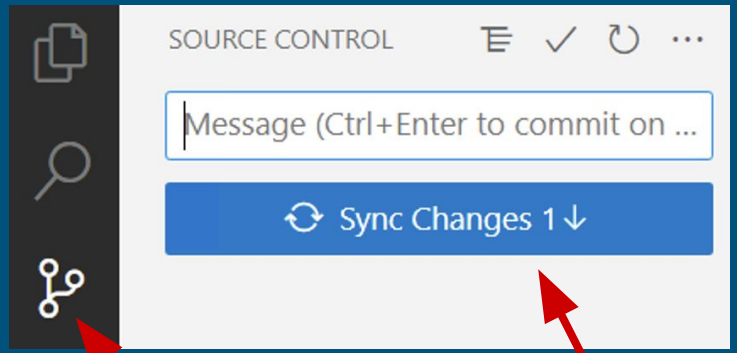


Terminal → New Terminal

```
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>
```

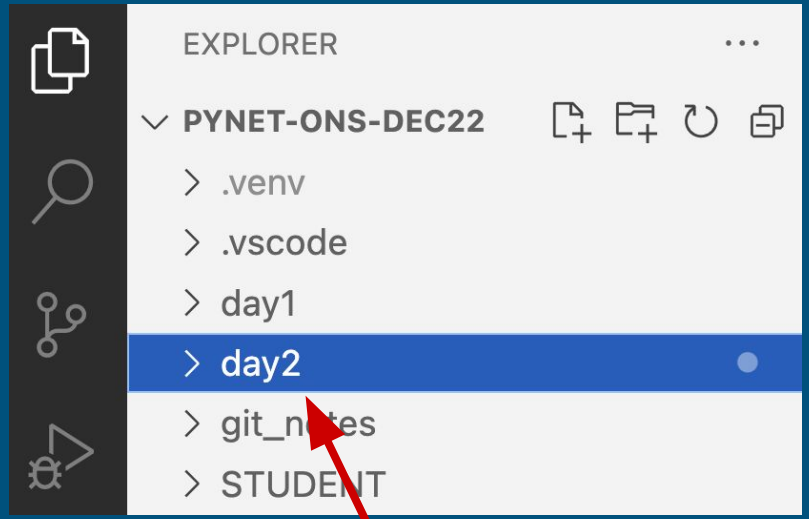
Virtual Environment Active

Environment Check



Select Git

Sync Changes



Should have a day2 folder.

Python Review

input/print

Assignment

Numbers

f-Strings

Variable Names

Lists

Strings

String Methods

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

- Create two lists named list1 and list2 containing various strings, integers, and floats.
- Create a list3 by concatenating list1 and list2 together.
- Use rich.print to print out this list3.
- Instead of using list concatenation now directly modify list1 by using list1.extend(list2)
- 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.

Back to 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']
```

String Slices - You Can Slice Strings Also

Same slicing rules apply: first index included, second index excluded.

```
In [1]: ip_addr = "192.168.223.77"

In [2]: ip_addr[:3]
Out[2]: '192'
```

```
In [4]: ip_addr[:-3]
Out[4]: '192.168.223'
```

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:
./day2/py_lists/lists_ex3.txt

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

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

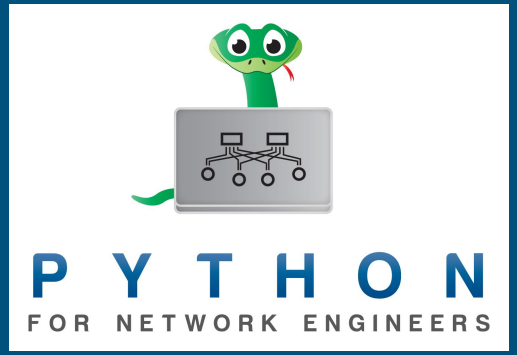
2. Use the split() method to create a new variable containing a list of the words in the sentence.

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

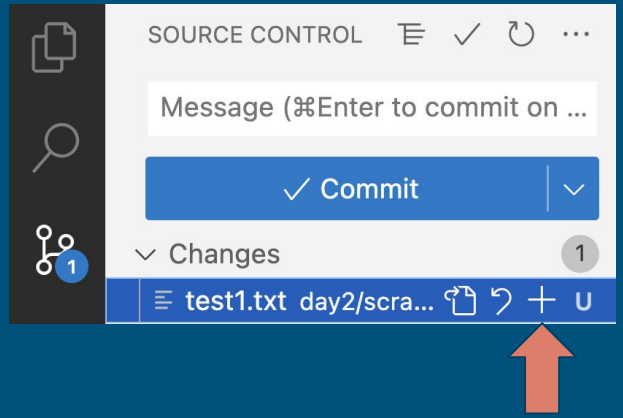
4. Change the last element of your newly created list to be just "words" i.e. no period at the end.

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

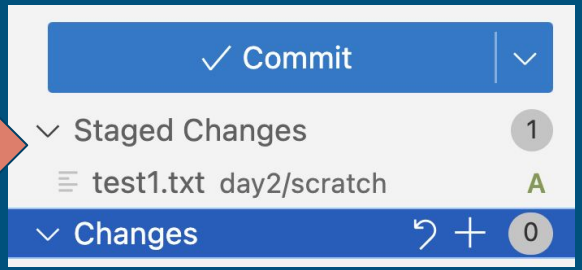
Review Git in VS Code - Adding File



Git Adding a File

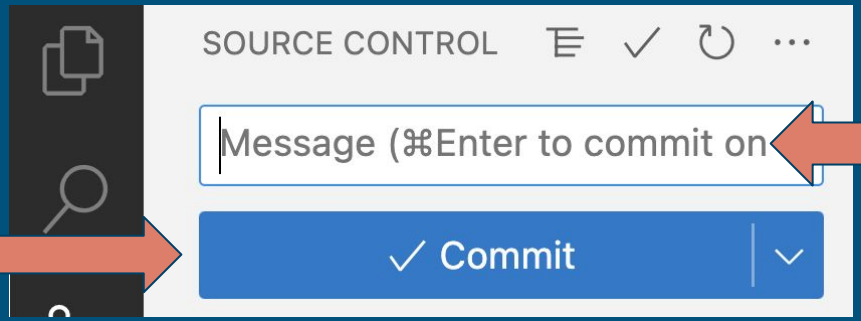


Staged Changes



Review Git in VS Code - Commit

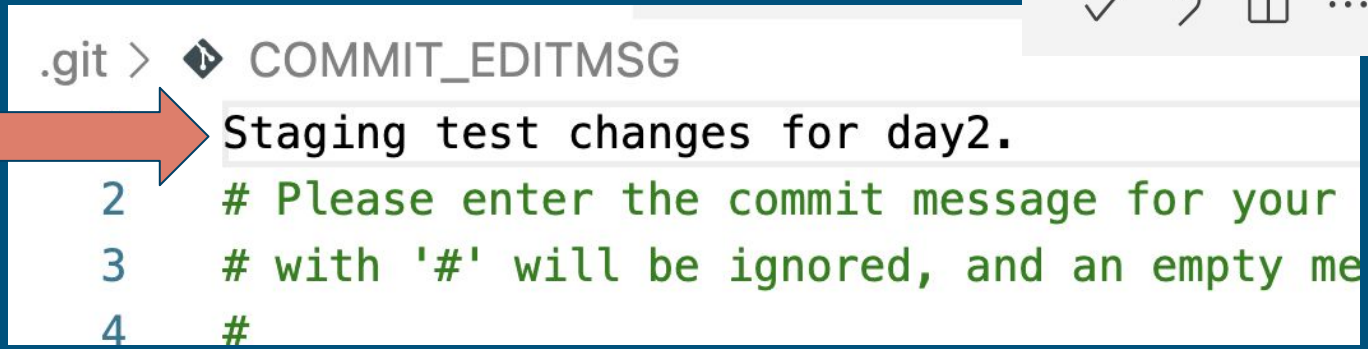
No message here will open up a file to submit a longer message.



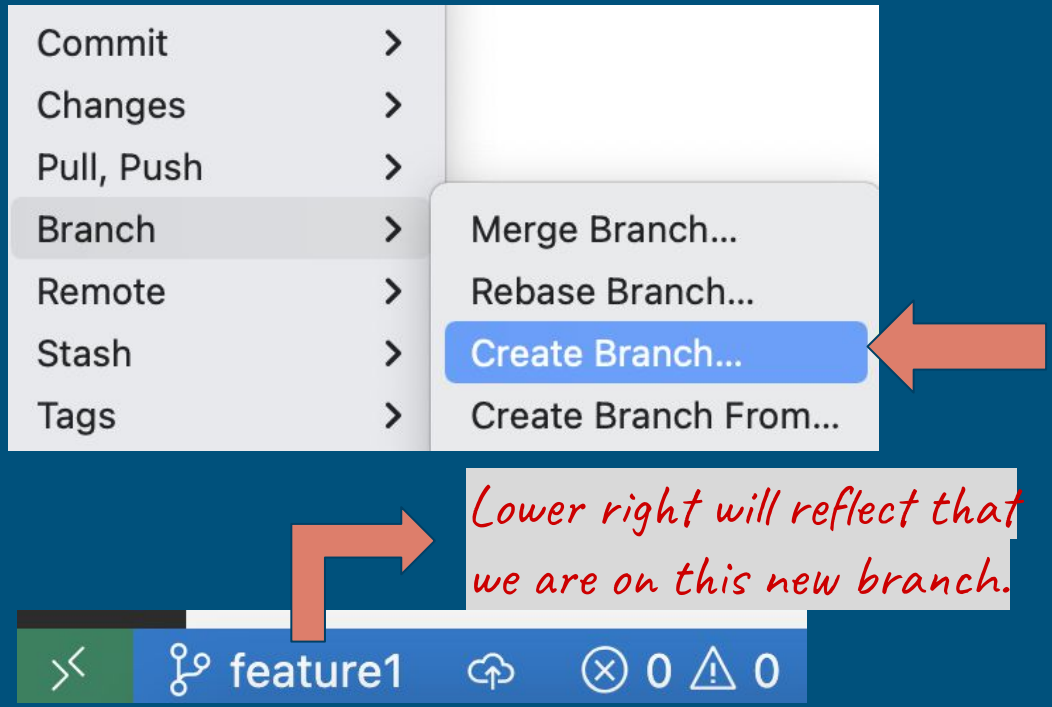
Commit

Click Check

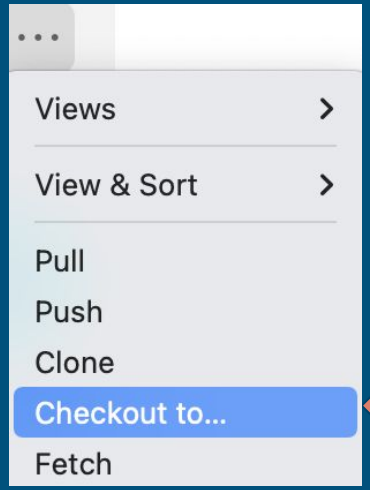
Our Message Here.



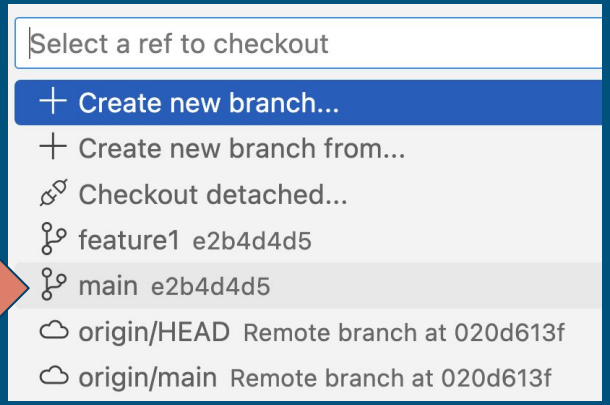
Review Git in VS Code - Create Branch.



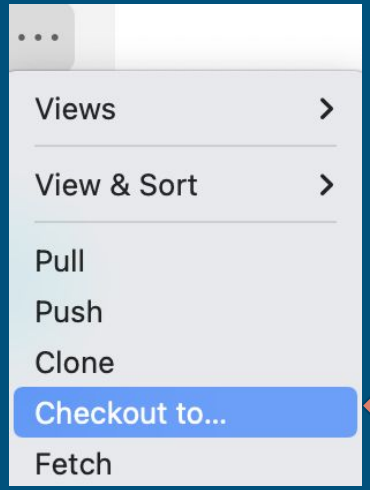
Review Git in VS Code - Switching Branches.



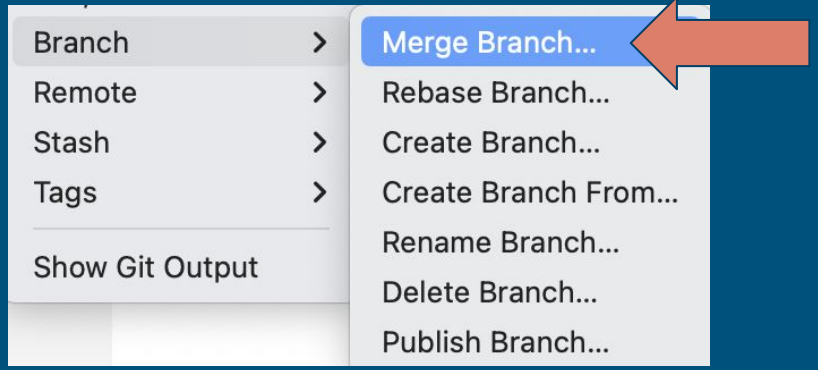
Select Checkout to...



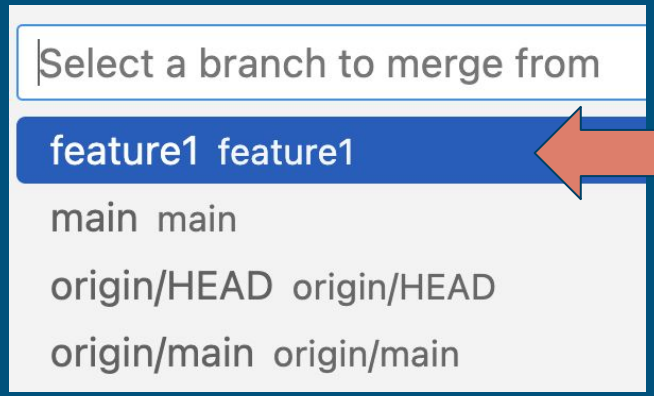
Review Git in VS Code - Merge Process



Select Checkout to...



Pick the branch that you want to merge into.



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:
./day2/py_files/files_ex1.txt

Booleans

```
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")
```

True or False – not strings, booleans are there own type.

```
In [7]: var1 = True

In [8]: type(var1)
Out[8]: bool
```

Booleans

```
In [9]: var1
Out[9]: True

In [10]: var2
Out[10]: False

In [11]: var1 and var2
Out[11]: False

In [12]: var1 or var2
Out[12]: True
```

Logical and - both must be True

Logical or - either variable must be True

```
In [13]: var2
Out[13]: False

In [14]: not var2
Out[14]: True
```

Logical complement.

Truish

```
In [18]: if "some string":
    ...:     print("Hello")
    ...:
```

```
In [19]: if 22:
    ...:     print("Hello")
    ...:
```

What happens if we use a non-boolean as a conditional expression?

Truish

```
In [20]: bool("some string")
Out[20]: True

In [21]: bool("")
Out[21]: False
```

Every other string is True.

Null-string is False.

```
In [22]: bool(22)
Out[22]: True

In [23]: bool(0)
Out[23]: False
```

For each data type, Python defines some value as the zero or null-value and treats that as False (when evaluated in a boolean context).

```
In [24]: bool(["element"])
Out[24]: True

In [25]: bool([])
Out[25]: False
```

Every other value is True

None

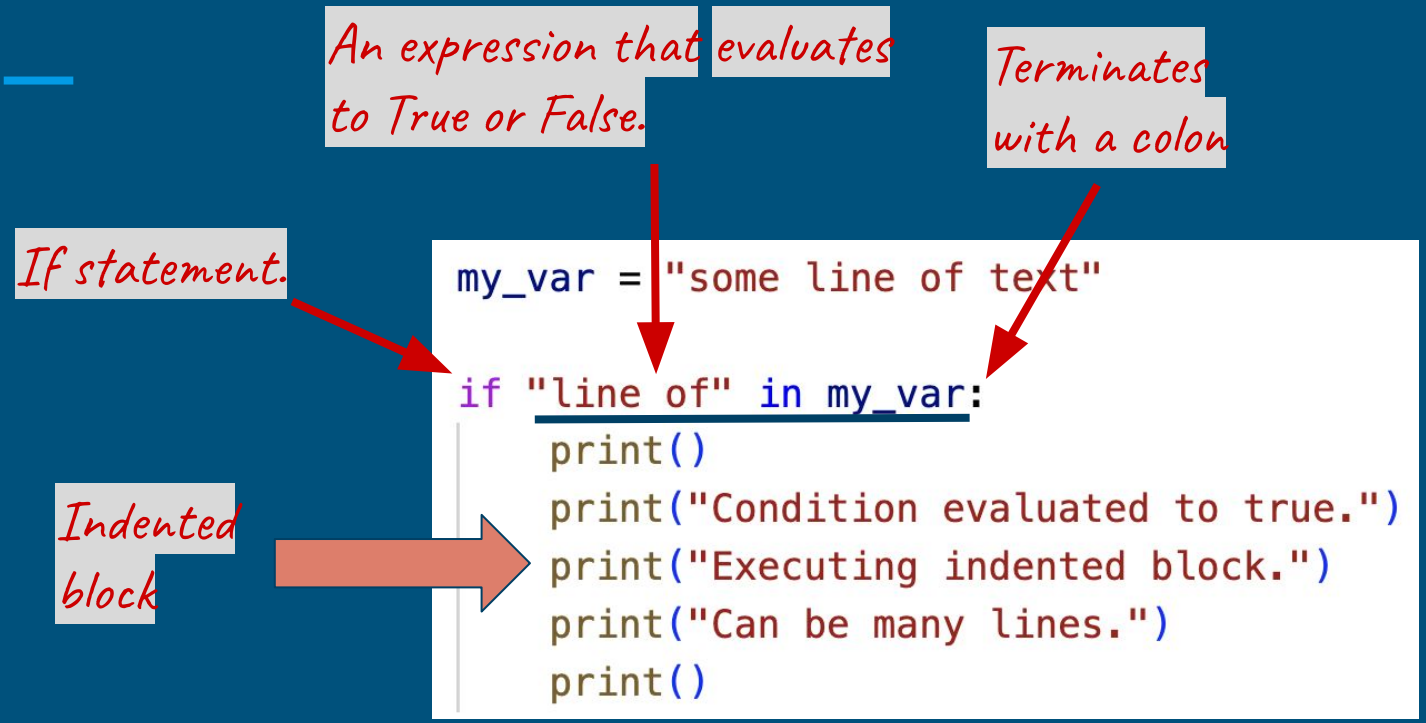
```
In [26]: var1 = None

In [27]: type(var1)
Out[27]: NoneType
```

Python has a "null" value named None which is its own type.

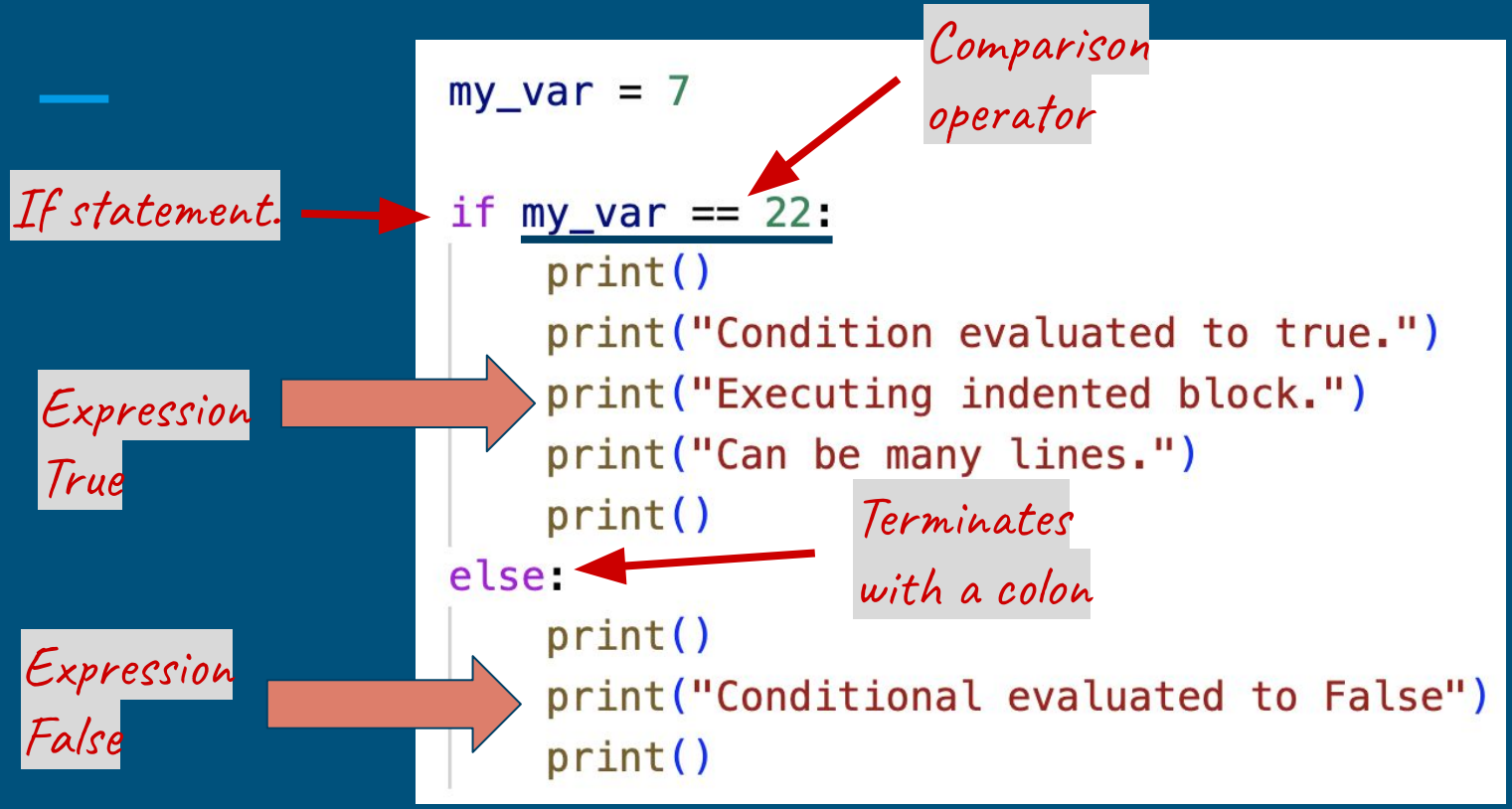
```
In [28]: bool(var1)
Out[28]: False
```

Unsurprisingly, if you evaluate None in a boolean context, it returns False.

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:
./day2/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:
./day2/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) */

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

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

For Loops

Similar to a "foreach" in powershell.

```
In [6]: octets
Out[6]: ['192', '168', '223', '77']

In [7]: for entry in octets:
   ...:     print(entry)
   ...:
192
168
223
77
```

Each time through the loop "entry" will get one of the elements of the "octets" list (one after the other).

For Loops

```
In [10]: sentence = "It was the best of times, it was the worst of times"

In [11]: words = sentence.split()

In [12]: for word in words:
    ...:     print(word)
    ...:
It
was
the
best
of
```

Let's break out of this loop.

Create a list of the words in the sentence.

Loop over the list of words.

If the word "best" is ever encountered, immediately end the loop.

```
In [22]: sentence
Out[22]: 'It was the best of times, it was the worst of times'

In [23]: words = sentence.split()

In [24]: for word in words:
    ...:     if "best" == word:
    ...:         break
    ...:     print(word)
    ...:
It
was
the
```

Continue Instead

If the word "best" is ever encountered, immediately jump back to the top of the for-loop (and work on the next entry).

"best" is missing (i.e. skipped over)

```
In [25]: for word in words:
    ...:     if "best" == word:
    ...:         continue
    ...:     print(word)
    ...:
It
was
the
of
times,
it
was
the
worst
of
times
```

Enumerate - What if I need the indices.

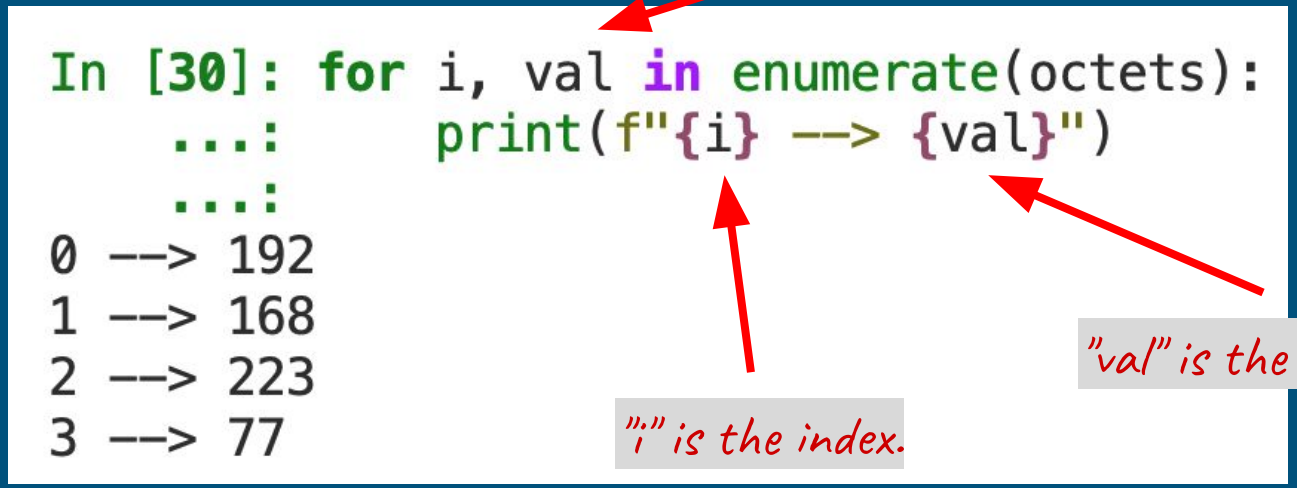
```
In [27]: octets
Out[27]: ['192', '168', '223', '77']

In [28]: for entry in enumerate(octets):
    ...:     print(entry)
    ...:
(0, '192')
(1, '168')
(2, '223')
(3, '77')
```

"enumerate" returns a two-element tuple.

The second element is the value of the list at that index.

The first element is the index.

Enumerate coupled with unpacking.



Immediately unpack what is returned by enumerate.

"i" is the index.

"val" is the corresponding value.

Exercise1: For Loops

1. Create a list containing the values 1-49, hint: list(range(1,50))

2. Loop over this list, printing each value

3. Use continue to skip over 13 (not print it)

4. Break when you hit 39 (stop printing after this and exit the loop).

Exercises:
./day2/py_loops/loops_ex1.txt

Exercise:

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
```

While Loops

```
In [2]: i = 1

In [3]: while i <= 10:
   ...:     print(i)
   ...:     i += 1
   ...:
1
2
3
4
5
6
7
8
9
10
```

Some expression: keep doing the loop "while" this thing is True.

You have to ensure that your loop eventually ends.

Keep looping until event.

Stay in the loop indefinitely.

```
In [4]: i = 1

In [5]: while True:
   ...:     print(i)
   ...:     if i == 10:
   ...:         break
   ...:     i += 1
   ...:
1
2
3
4
5
6
7
8
9
10
```

Break out of loop upon this event.

Continue still works (as expected)

```
In [8]: i = 1

In [9]: while True:
   ...:     if i == 5:
   ...:         i += 1
   ...:         continue
   ...:     print(i)
   ...:     if i == 10:
   ...:         break
   ...:     i += 1
   ...:
1
2
3
4
6
7
8
9
10
```

Don't print if 5 (i.e. jump back to the top of the while loop).

Still need to make sure that we end the loop.

Exercise2: While Loops

1. Use a while loop and a counter variable i.
2. Initialize i to an initial value of 0
3. Stay in the while loop until i is greater than 49
4. Increment i each time through the while loop
5. Print out i (except don't print out i when it is 13).

Exercises:
./day2/py_loops/loops_ex2.txt

Dictionaries

You use curly-braces to declare a dictionary.

```
In [10]: person = {
    ...:   "name": "Kirk Byers",
    ...:   "address": "100 Whatever Lane",
    ...:   "city": "San Francisco",
    ...:   "state": "CA",
    ...:   "zip_code": "94105",
    ...:   "phone_number": "415-867-5309"
    ...: }
```

Key-Value Pairs

AKA: hashes or hash-maps

Dictionaries - Accessing Values.

```
In [11]: person["name"]
Out[11]: 'Kirk Byers'

In [12]: person["zip_code"]
Out[12]: '94105'
```

Back to square-brackets!

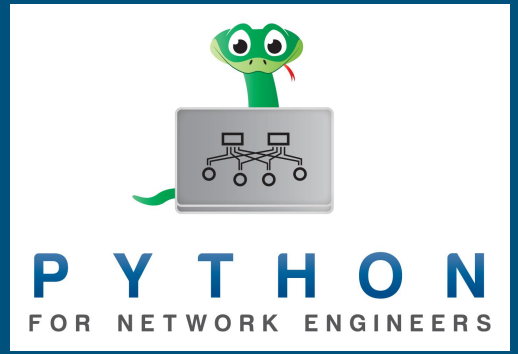
Provide the key; retrieve the value.

Lookup is very fast-even for very large dictionaries: O(1) – i.e. on average the lookup will take the same time regardless of dictionary size.

Dictionaries - Accessing a Key that doesn't exist.

```
In [16]: person["country"]
---------------------------
KeyError
Cell In[16], line 1
----> 1 person["country"]

KeyError: 'country'
```

We get back a KeyError exception.



Dictionaries - Accessing a Key that doesn't exist using get()

Use the "get()" method.

```
In [17]: result = person.get("country")

In [18]: print(result)
None
```

If the given key doesn't exist, then don't generate an Exception instead just return None.

Dictionaries - Accessing a Key that doesn't exist using get()

Can change the default value that is returned from None to something else.

```
In [3]: result = person.get("country", "US")

In [4]: print(result)
US
```

Dictionaries - We can also .pop() keys

```
In [22]: ip_addresses = {
    ...:   "sf": "10.18.22.1",
    ...:   "la": "10.19.1.1",
    ...:   "chi": "10.20.100.1",
    ...:   "nyc": "10.21.254.1"
    ...: }

In [23]: sf_addr = ip_addresses.pop("sf")

In [24]: sf_addr
Out[24]: '10.18.22.1'

In [25]: ip_addresses
Out[25]: {'la': '10.19.1.1', 'chi': '10.20.100.1', 'nyc': '10.21.254.1'}
```

pop() using the key

The corresponding value is returned.

And removed from the dictionary.

Dictionaries - Looping over dictionaries.

```
In [28]: ip_addresses = {
    ...:   "sf": "10.18.22.1",
    ...:   "la": "10.19.1.1",
    ...:   "chi": "10.20.100.1",
    ...:   "nyc": "10.21.254.1"
    ...: }

In [29]: for k in ip_addresses:
    ...:     print(k)
    ...:
sf
la
chi
nyc
```

What do we get back when loop over a dictionary?

Dictionaries - Are dictionaries inherently ordered?

```
In [30]: ip_addresses = {
    ...:   "sf": "10.18.22.1",
    ...:   "la": "10.19.1.1",
    ...:   "chi": "10.20.100.1",
    ...:   "nyc": "10.21.254.1"
    ...: }

In [31]: ip_addresses["wdc"] = "10.22.2.1"

In [32]: for k in ip_addresses:
    ...:     print(k)
    ...:
sf
la
chi
nyc
wdc
```

They are now...as of Python 3.7 dictionaries are ordered based on insertion order.

Dictionary (loops) - What if I want the values and not the keys?

```
In [33]: ip_addresses
Out[33]:
{'sf': '10.18.22.1',
 'la': '10.19.1.1',
 'chi': '10.20.100.1',
 'nyc': '10.21.254.1',
 'wdc': '10.22.2.1'}

In [34]: for v in ip_addresses.values():
    ...:     print(v)
    ...:
10.18.22.1
10.19.1.1
10.20.100.1
10.21.254.1
10.22.2.1
```

Use the .values() method

Dictionary (loops) - What if I want both the key and the value?

```
In [35]: ip_addresses
Out[35]:
{'sf': '10.18.22.1',
 'la': '10.19.1.1',
 'chi': '10.20.100.1',
 'nyc': '10.21.254.1',
 'wdc': '10.22.2.1'}

In [36]: for k, v in ip_addresses.items():
    ...:     print(f"{k} --> {v}")
    ...:
sf --> 10.18.22.1
la --> 10.19.1.1
chi --> 10.20.100.1
nyc --> 10.21.254.1
wdc --> 10.22.2.1
```

Immediate unpacking into two separate variables: k and v.

Use the .items() method

Exercise1: Dictionaries

Exercises:
./day2/py_dict/dict_ex1.txt

a. Create a dictionary representing a network device.

b. Assign it an ip address, a username, a password, a vendor, and a model field.

c. Loop over this dictionary printing out all of the keys, and values

d. Update the password to be a new value

e. Add a secret field to the dictionary

f. Use the .get() method to try to retrieve a non-existent ‘device_type’ field. Return a default value of ‘cisco_ios’ when the .get() key lookup fails.

Nesting Dictionaries

Outermost Key

```
my_devices = {
    "sf-rtr1": {
        "hostname": "cisco3.lasthop.io",
        "device_type": "cisco_xe",
        "username": "admin",
        "password": "bogus123"
    },
    "chi-rtr1": {
        "hostname": "cisco3.lasthop.io",
        "device_type": "cisco_xe",
        "username": "admin",
        "password": "bogus123"
    }
}
```

Inner dictionary is the corresponding value.

Nesting Dictionaries

Access the outer key gives us the inner dictionary.

```
In [2]: my_devices["chi-rtr1"]
Out[2]:
{'hostname': 'cisco3.lasthop.io',
 'device_type': 'cisco_xe',
 'username': 'admin',
 'password': 'bogus123'}
```

Nesting Dictionaries

```
In [2]: my_devices["chi-rtr1"]
Out[2]:
{'hostname': 'cisco3.lasthop.io',
 'device_type': 'cisco_xe',
 'username': 'admin',
 'password': 'bogus123'}

In [3]: my_devices["chi-rtr1"]["hostname"]
Out[3]: 'cisco3.lasthop.io'

In [4]: my_devices["chi-rtr1"]["username"]
Out[4]: 'admin'
```

Then can drill into key-value pairs of the inner dictionary. Once again read left-to-right.

Dictionary Keys Can Also Refer to Lists.

```
In [13]: print(my_dict)
{
    'key1': ['some', 'list', 'of', 'strings'],
    'key2': 22,
    'key3': 'whatever',
    'key4': 'something else'
}
```

Use "key1" to retrieve the list.

```
In [14]: my_dict["key1"]
Out[14]: ['some', 'list', 'of', 'strings']

In [15]: my_dict["key1"][-1]
Out[15]: 'strings'
```

Retrieve the last element of the list.

Exercise2: Nesting Dictionaries

a. Create two devices with the same key-value pairs as specified in the first dictionary exercise (dict_ex1). These two devices should be assigned to variables named: chi_rtr1, and chi_rtr2 respectively.

b. Create a new blank dictionary named net_devices.

c. Assign the two devices created in step-a to the net_devices dictionary using the keys "chi_rtr1" and "chi_rtr2" respectively (in other words create a nested dictionary).

d. Use rich.print to print out your net_devices dictionary.

Exercises:
./day2/py_dict/dict_ex2.txt

Exercise: Strings and Dictionaries

Expand on "./day2/py_strings/strings_ex1.txt" 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

Exception Handling

Gracefully handle errors.


```
In [2]: my_dict["no_key"]
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-2-56d0da37b330> in <module>
----> 1 my_dict["no_key"]

KeyError: 'no_key'
```

```
In [3]: my_list = []

In [4]: my_list[7]
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-4-352a83797fff> in <module>
----> 1 my_list[7]

IndexError: list index out of range
```

Exception Handling

Gracefully handle errors.

```
In [5]: for i in range(10):
   ...:     print(i)
   ...:      print("Indentation off")
  File "<ipython-input-5-ceba4aa1ecbd>", line 3
    print("Indentation off")
    ^
IndentationError: unexpected indent
```

```
In [6]: forx i in range(10):
   ...:     print(i)
  File "<ipython-input-6-dec1f37c6ba2>", line 1
    forx i in range(10):
         ^
SyntaxError: invalid syntax
```

Exception Handling

Gracefully handle errors.


```
In [7]: "hello" + 12
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-7-e8992ec33927> in <module>
----> 1 "hello" + 12

TypeError: can only concatenate str (not "int") to str
```

```
In [8]: open("bogus_file.txt")
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-8-a6a0a3b54d04> in <module>
----> 1 open("bogus_file.txt")

FileNotFoundError: [Errno 2] No such file or directory: 'bogus_file.txt'
```

Exception Handling

Actually handling the exception - this particular error might happen here.

```
my_ds = {}
try:
    # An error might happen here
    my_ds["invalid_key"]
except KeyError:
    # The specified error happened - what do I do about it.
    print("Handling KeyError exception")
```

Exception Handling

```
my_list = []
try:
    # Another error might happen
    my_list[7]
except IndexError:
    # The error happened - handle it
    print("The given list index didn't exit")
```

Handling Generic Exceptions

Be careful - you could be hiding errors!

```
my_ds = {}
my_list = []
try:
    # An error might happen here
    my_ds["invalid_key"]
    my_list[7]
except Exception:
    # An error happened - keep going.
    print("Generic exception handling")
```

Finally: Do Something in Either Case

```
my_ds = {}
my_list = []
try:
    # An error might happen here
    my_ds["invalid_key"]
    my_list[7]
except Exception:
    # An error happened - keep going.
    print("Generic exception handling")
finally:
    print("Error or no error - print this message out")
```

You can "raise" your own exceptions.

```python
my_ip = "192.168.1.1"
if my_ip != "10.1.1.1":
    raise ValueError(f"You are connecting to the wrong device:\n\n{my_ip}")
```

```
(.venv) [ktbyers@pydev2 EP]$ python invalid_ip.py
Traceback (most recent call last):
  File "/home/ktbyers/EP/invalid_ip.py", line 3, in <module>
    raise ValueError(f"You are connecting to the wrong device:\n\n{my_ip}")
ValueError: You are connecting to the wrong device:

192.168.1.1
```

Exceptions - Exercise1

a. Create a dictionary representing a network device

b. Assign it an ip address, a username, a password, a vendor, and a model field.

c. Try to retrieve the ‘device_type’ just by directly accessing the key (a KeyError exception will occur)

d. Gracefully, handle the exception and print the message that the device_type field is not found.

GitHub: {{ repo }}/day2/py_except/except_dict_ex1.txt

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

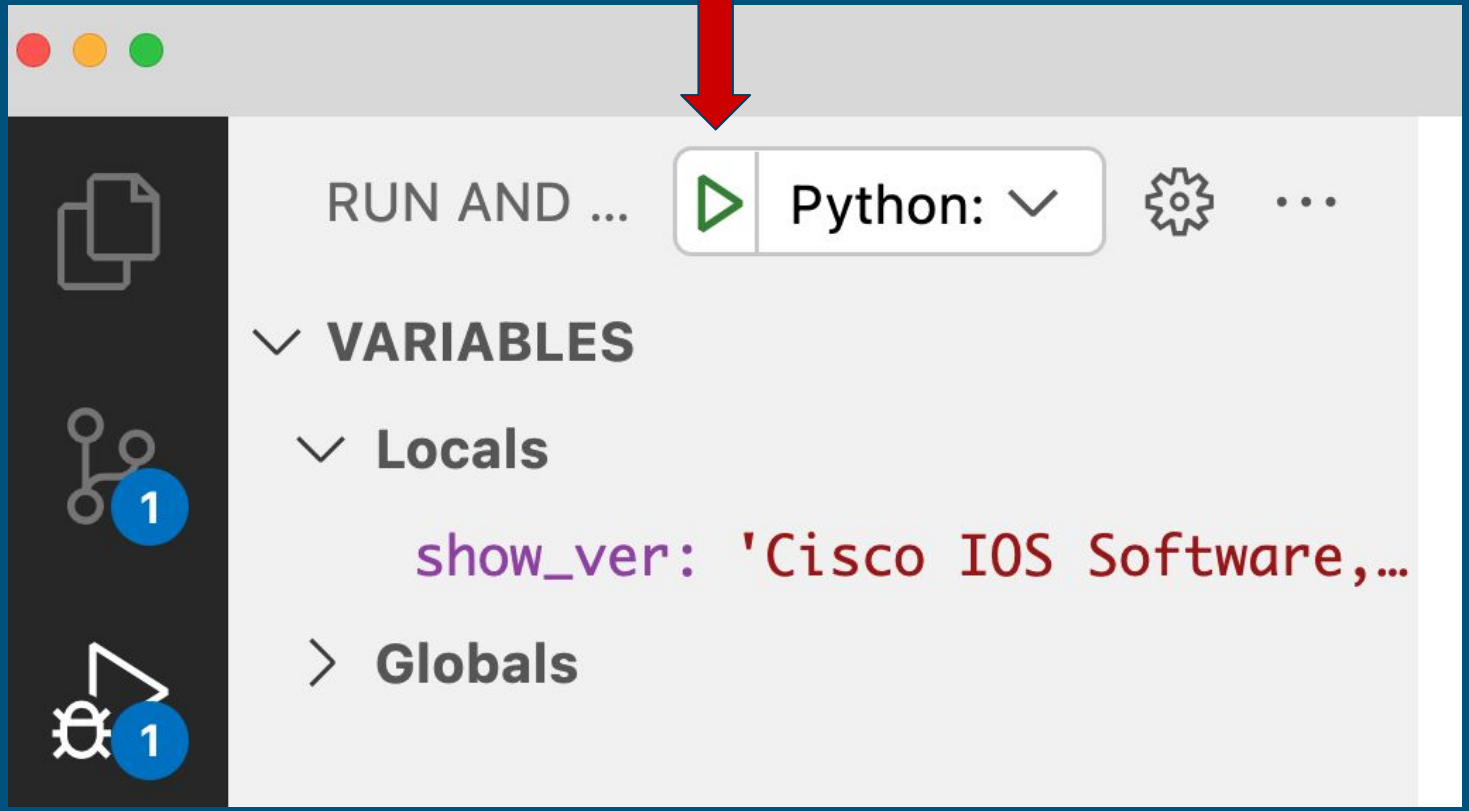
VS Code Debugging

```
5
6   def find_serial_number(show_ver):
7       serial_number = ""
8       for line in show_ver.splitlines():
9           if "Processor board ID" in line:
```

Setting a breakpoint

Executing the Debugger

Run

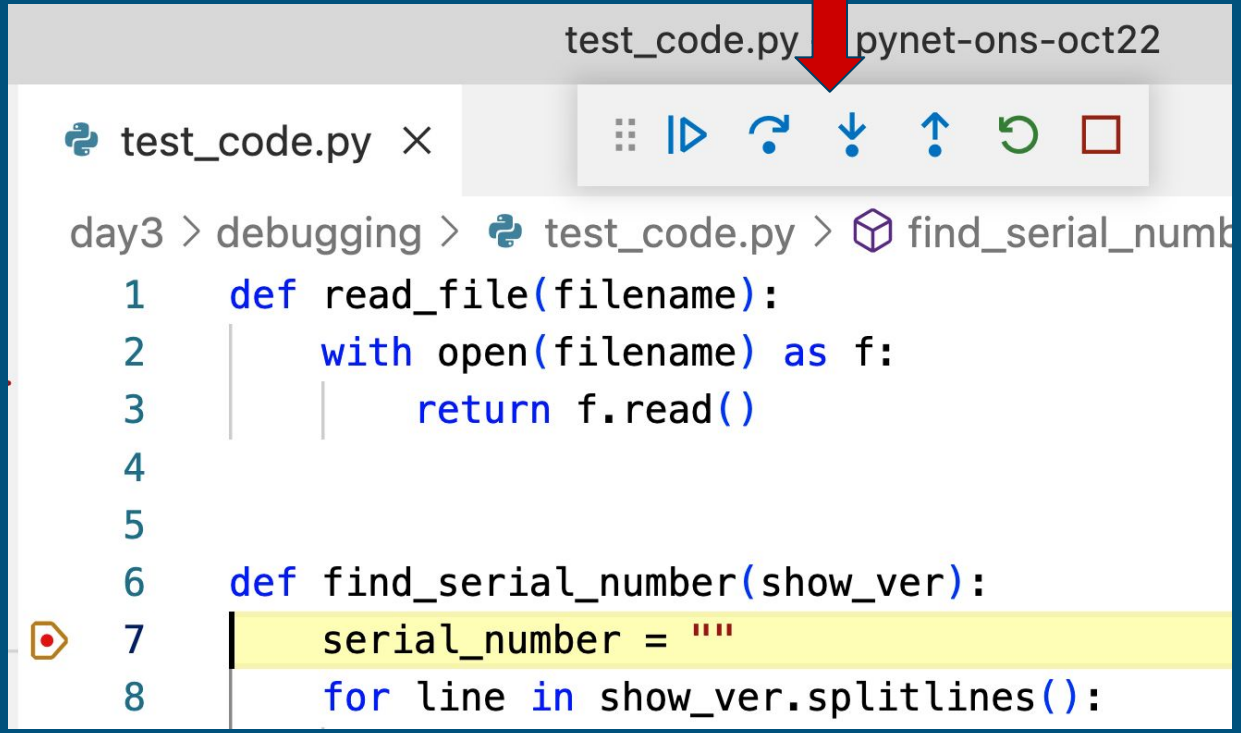


Run & Debug

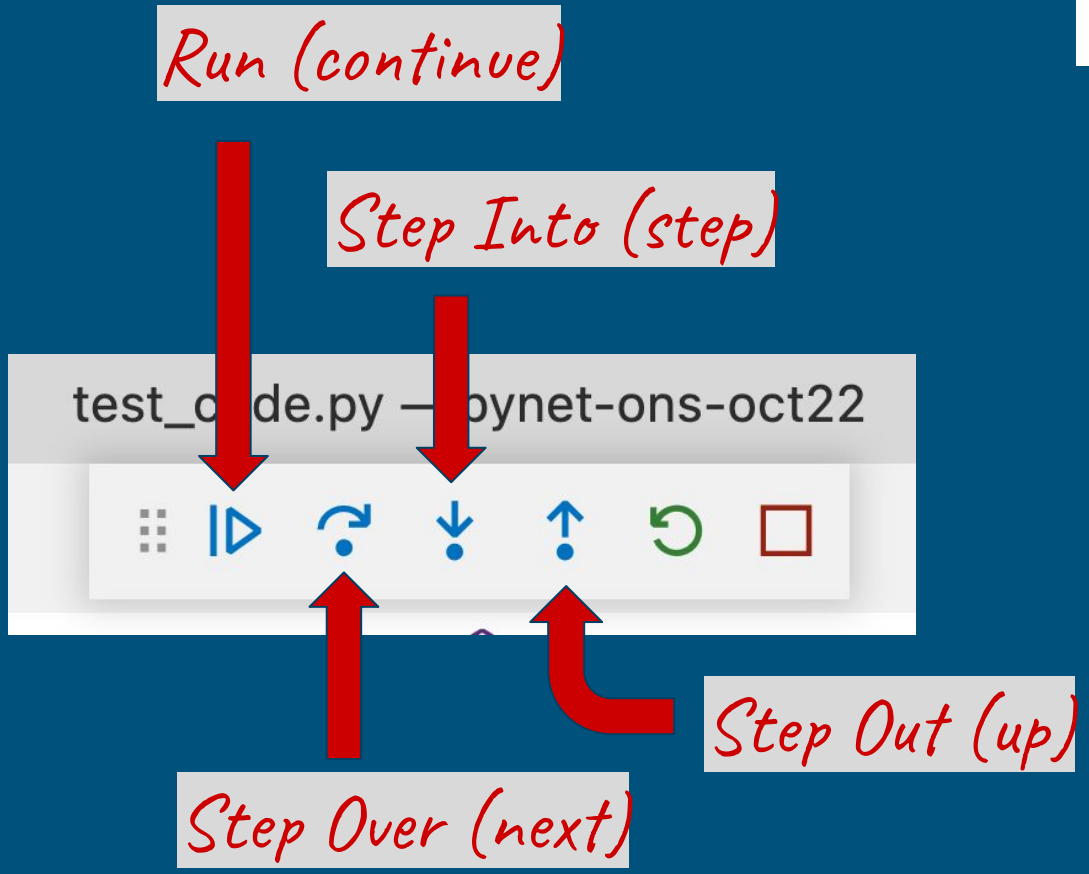
Executing the Debugger

Debugger Toolbar

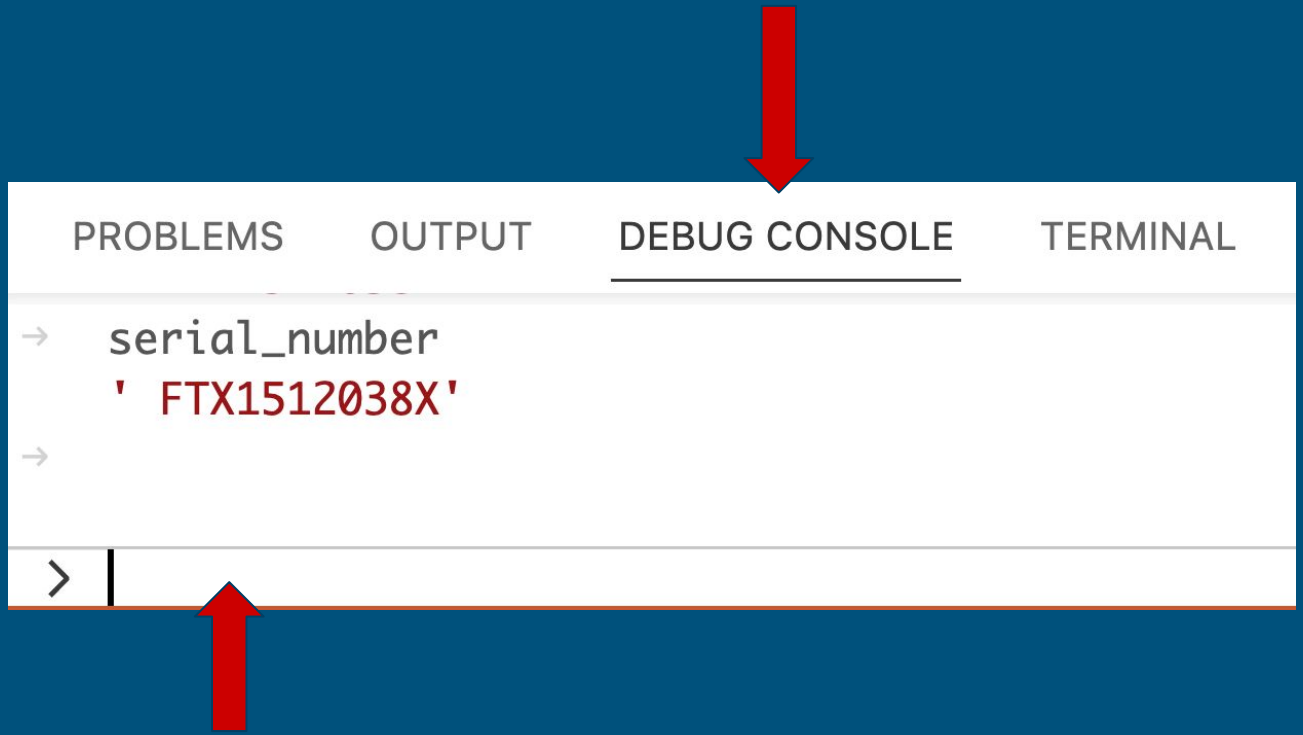
Current Location



Debugger Toolbar



Debug Console



Inspect variables

Exercise

Exercises:
./day2/review_exercises/review_ex1.txt

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.

4. Using the Python debugger set a breakpoint in your code and inspect the two dictionaries before they are printed to the screen. Use step over to step a few lines through the code. Add a second breakpoint and use continue to run until this second breakpoint is encountered.

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

Libraries: Two Different Import Formats

Find the "re" library

```
In [4]: import re

In [5]: re.search("pattern", "some string")
```

Names must be prefixed with "re."

Process this entire file-line by line.

Here is where Python found "re"

```
In [6]: re.__file__
Out[6]: '/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/re.py'
```

Libraries: Two Different Import Formats

Still finds the "re" library

```
In [1]: from re import search

In [2]: search("pattern", "some string")
```

But this form does not require prefixing the name with "re.".

Still processes the entire file–line by line.

Import in this form processes the file in the same way; it just changes the name references in your program.

```
In [3]: from re import search as my_search
```

sys.path and $PYTHONPATH


```
import sys
from rich import print

print(sys.path)
```

How does Python find things?

```
# 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.

$PYTHONPATH and VS Code

```
    "python.terminal.executeInFileDir": true,
    "terminal.integrated.env.osx": {
        "PYTHONPATH": "${workspaceFolder}/src"
    },
    "terminal.integrated.env.windows": {
        "PYTHONPATH": "${workspaceFolder}/src"
    }
}
```

```
(.venv) $ env | grep PYT
PYTHONPATH=/Users/ktbyers/GIT/pynet-ons-oct22/src
(.venv) $
```

$PYTHONPATH and VS Code

On the road to reusable code.

We can create a Python file and locate it in {workspace}/src

```
(.venv) $ cat src/test_code.py

def my_func():
    print("Hello")
```

We can import and use it (from anywhere on our system).

```
In [1]: from test_code import my_func

In [2]: my_func()
Hello
```

PIP - Package Installer for Python.


```
(.venv) $ python -m pip list
Package            Version
------------------ -------
appnope            0.1.3
astroid            2.12.11
```

pypi = Python Package Index

```
(.venv) $ python -m pip show rich
Name: rich
Version: 12.6.0
Summary: Render rich text, tables, progress bars, syntax highlighting, markdown and more to the t
nal
Home-page: https://github.com/willmcgugan/rich
Author: Will McGugan
Author-email: willmcgugan@gmail.com
License: MIT
Location: /Users/ktbyers/GIT/pynet-ons-oct22/.venv/lib/python3.10/site-packages
Requires: commonmark, pygments
Required-by: pdbr
```

PIP - Package Installer for Python.


```
(.venv) $ python -m pip uninstall rich
Found existing installation: rich 12.6.0
Uninstalling rich-12.6.0:
  Would remove:
    /Users/ktbyers/GIT/pynet-ons-oct22/.venv/lib/python3.10/site-packages/rich-12.6.0.dist-info/*
    /Users/ktbyers/GIT/pynet-ons-oct22/.venv/lib/python3.10/site-packages/rich/*
Proceed (Y/n)? y
  Successfully uninstalled rich-12.6.0
```

```
(.venv) $ python -m pip install rich==12.6.0
Collecting rich==12.6.0
  Using cached rich-12.6.0-py3-none-any.whl (237 kB)
Requirement already satisfied: pygments<3.0.0,>=2.6.0 in ./.venv/lib/python3.10/site-packages (fr
ich==12.6.0) (2.13.0)
Requirement already satisfied: commonmark<0.10.0,>=0.9.0 in ./.venv/lib/python3.10/site-packages
m rich==12.6.0) (0.9.1)
Installing collected packages: rich
Successfully installed rich-12.6.0
```

PIP - Package Installer for Python.

```
(.venv) $ python -m pip freeze
appnope==0.1.3
astroid==2.12.11
asttokens==2.0.8
backcall==0.2.0
black==22.10.0
```

PIP - Package Installer for Python.


```
(.venv) $ python -m pip install -r ./requirements-dev.txt
Requirement already satisfied: ipython in ./.venv/lib/python3.10/site-packages
s-dev.txt (line 1)) (8.5.0)
Requirement already satisfied: pdbr in ./.venv/lib/python3.10/site-packages (fr
ev.txt (line 2)) (0.7.3)
```

```
(.venv) [ktbyers@pydev2 netmiko]$ python -m pip install -e .
Obtaining file:///home/ktbyers/netmiko
  Preparing metadata (setup.py) ... done
Requirement already satisfied: setuptools>=38.4.0 in ./.venv/lib/pyt
```

```
(.venv) [ktbyers@pydev2 netmiko]$ pip list | grep netmiko
netmiko                       4.1.2     /home/ktbyers/netmiko
```

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

List Exercise4

Exercises:
./day2/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.

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,
}
```