Index

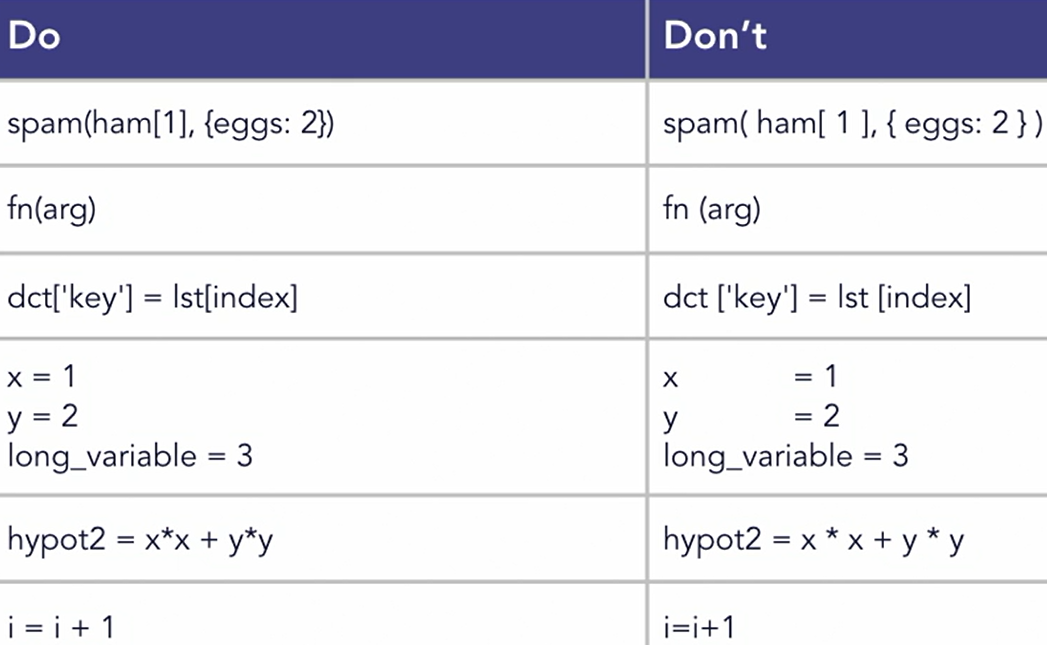
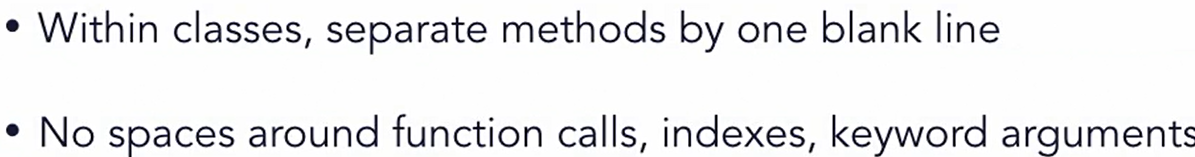
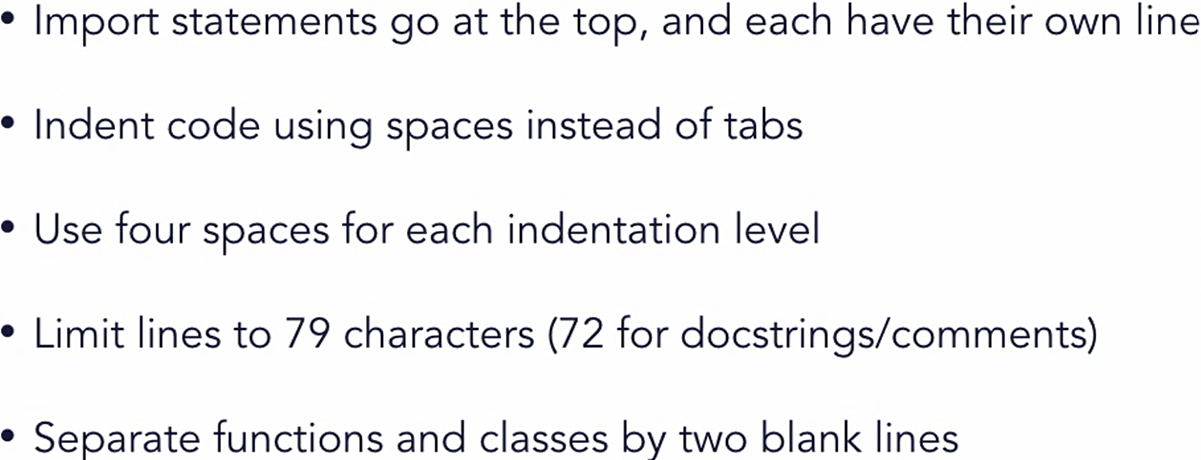
1. Lyn: 2019 Advanced Python
2. Lyn: Py Data Analysis [ skip ]
3. Lyn: Advanced Python Working with Databases
4. Lyn: 2019 Python Design Patterns
5. Lyn: 2019 Python Advanced Design Patterns
6. d

**Lyn: Advanced Python**

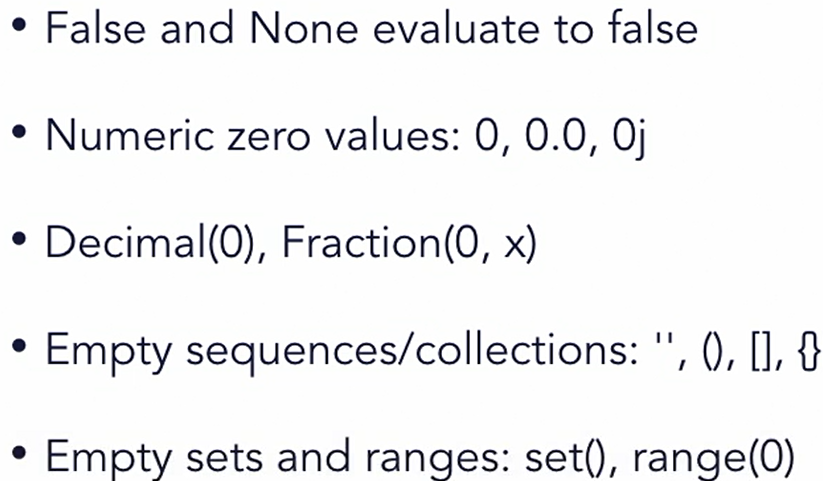
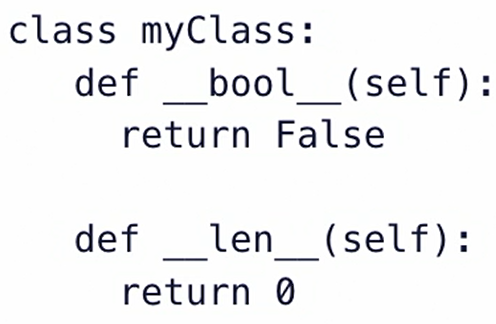
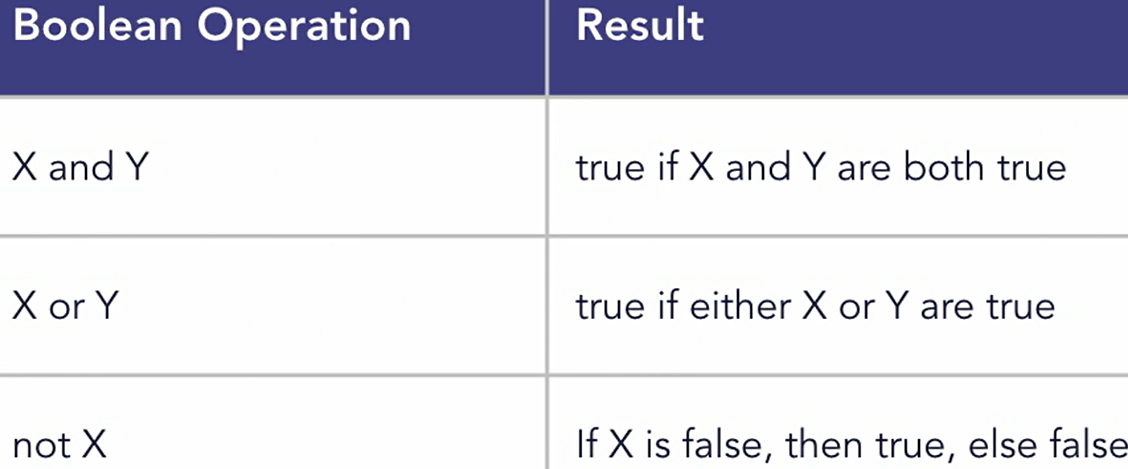
**// 101 Python code style**

PEP 8 – Styling Guide for Python Code

/peps/pep-0008



**// 102 Truth value testing**

All below are examples of False value in Python:

x = list(); print(bool(x)); # False

y = dict(); print(bool(y)); # False

**// 103 String VS Bytes**

# strings and bytes are not directly interchangeable

# strings contain unicode, bytes are raw 8-bit values

# define some starting values

b = bytes([0x41, 0x42, 0x43, 0x44]); print(b)

s = "This is a string"; print(s)

# Try combining them. This will cause an error:

# print(s+b) # TypeError: can only concatenate....

# Bytes and strings need to be properly encoded and decoded before you can work on them together

s2 = **b.decode('utf-8')**; print(s+s2)

b2 = **s.encode('utf-8');** print(b+b2)

b3 = **s.encode('utf-32')** # encode the string as UTF-32

**// 104 Template Strings**

from string import Template

# create a template with placeholders

**templ = Template("**You're watching ${title} by ${author}")

# use the substitute method with keyword arguments

str2 = **templ.substitute(title="Adv Python", author="Joe"**)

print(str2)

# use the substitute method with a dictionary

data = { "author": "Joe”, "title": " Python" }

str3 = **templ.substitute**(data); print(str3)

**// 201 Utilities** # Build-in functions

any, all, min, max, sum

**// 202 Iterator**

days = ["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"]

daysFr = ["Dim", "Lun", "Mar", "Mer", "Jeu", "Ven", "Sam"]

i = **iter(days);** print(**next(i)**); print(**next(i));** print(**next(i)**)

# iterate using a function and a sentinel

***# iter(object, sentinel)***

***# sentinel means on what value iter stop calling next()***

**with open("testfile.txt", "r") as fp:**

**for line in iter(fp.readline, ''): print(line)**

# using enumerate reduces code and provides a counter

**for i, m in enumerate(days, start=1):** print(i, m)

# use zip to combine sequences

for m in zip(days, daysFr): print(m)

**for i, m in enumerate(zip(days, daysFr), start=1):**

print(i, m[0], "=", m[1], "in French")

==== ====

# reading fixed-width blocks from a binary file until the end

from functools import partial

with open("testfile.txt", "r") as f:

for block in **iter(partial(f.read, 64), b''):** print(block)

**// 203 Transform** # use functions like **sorted, filter, map**

def filterFunc(x):

if x % 2 == 0: return False

return True

def squareFunc(x): return x\*\*2

def toGrade(x):

if (x >= 90): return "A"

elif (x >= 80 and x < 90): return "B"

elif (x >= 70 and x < 80): return "C"

elif (x >= 65 and x < 70): return "D"

return "F"

nums = (1, 8, 4, 5, 13, 26, 381, 410, 58, 47)

chars = "abcDeFGHiJklmnoP"

grades = (81, 89, 94, 78, 61, 66, 99, 74)

# use filter to remove items from a list

print([**filter(filterFunc, nums)])**

print([x for x in nums if x % 2 != 0])

# use map to create a new sequence of values

squares = list(**map(squareFunc, nums**)); print(squares)

# use sorted and map to change numbers to grades

grades = sorted(grades)

letters = list(**map(toGrade, grades));** print(letters)

**// 204 Itertools and Lambda function, lambda**

import itertools

def testFunction(x): return x < 40

# cycle iterator can be used to cycle over a collection

seq1 = [1, 2]; cycle1 = **itertools.cycle**(seq1)

print(next(cycle1)) # 1

print(next(cycle1)) # 2 print(next(cycle1)) # 1

# use count to create a simple counter

count1 = **itertools.count**(100, 10)

print(next(count1)) # 100

print(next(count1)) # 110 print(next(count1)) # 120

# accumulate creates an iterator that accumulates values

vals = [10,20,30,40,50,40,30]

# sum-of-digits [10, 30, 60, 100, 150, 190, 220]

print(list(**itertools.accumulate**(vals)))

acc = **itertools.accumulate(vals, max)**

print(list(acc)) # [10, 20, 30, 40, 50, 50, 50]

# use chain to connect sequences together

x = itertools.chain("ABCD", "1234"); print(list(x))

# return values until certain condition is met that stops them

print(list(**itertools.dropwhile(lambda x: x < 40, vals**)))

print(list(**itertools.takewhile(testFunction, vals)))**

**// 301 function documentation string from py terminal**

>>> print(map.\_\_doc\_\_)

>>> print(collections**.\_\_doc\_\_)**

>>> def myFunction(arg1, arg2=None):

"""myFunction(arg1, arg2=None) --> Doesn't really do anything special.

Parameters:

arg1: the first argument. Whatever you feel like passing.

arg2: the second arg Defaults to None. Whatever happy.

"""

print(arg1, arg2)

>>> print(myFunction.\_\_doc\_\_)

**# PEP 257**

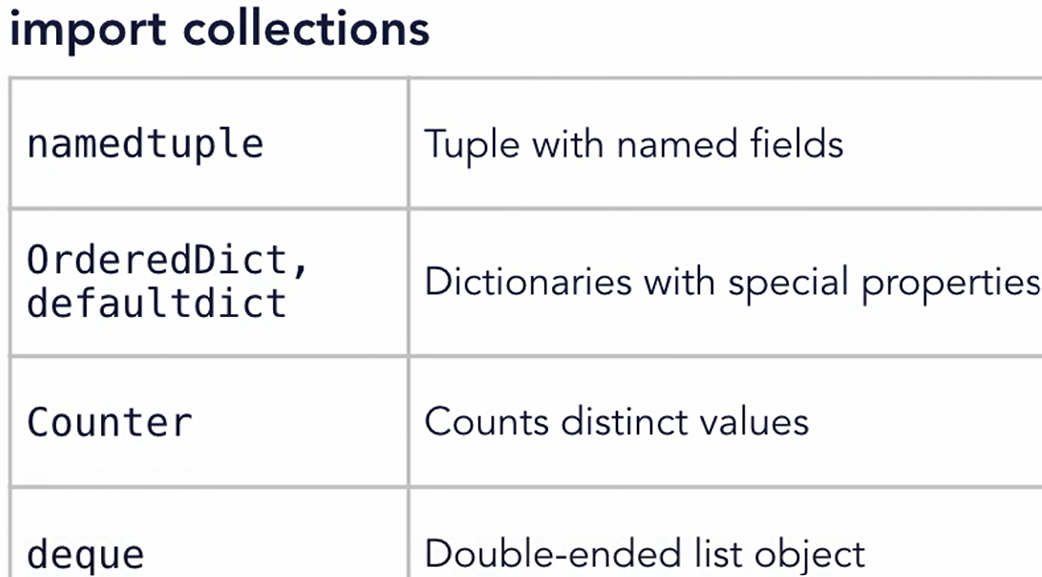
**// 302 Variable argument list**

def addition(base, \*args):

**// 303 Lambda function : Anonymous**

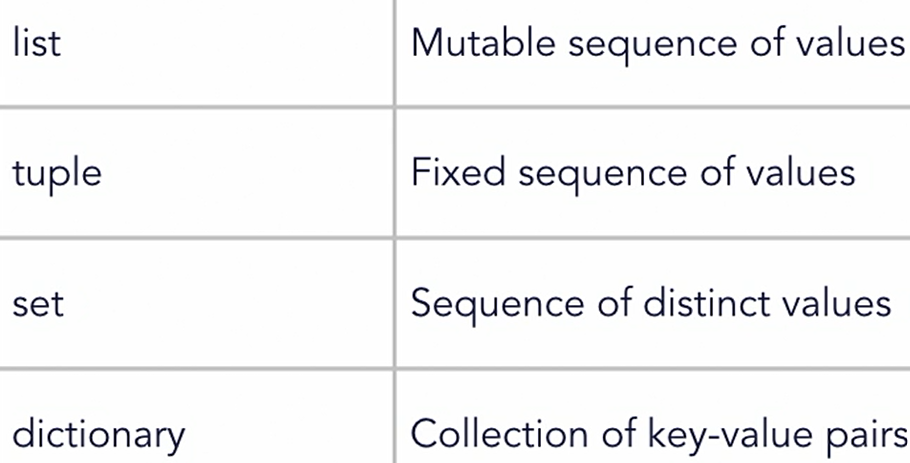
print(list**(map(lambda t:** (t-32) \* 5/9, ftemps)))

# ftemps is iterable list here

**// 304 Keyword only arguments**

def myFunction(arg1, arg2, \*, suppressExceptions=False):

# keyword arguments can come later after all the other args

**// 401 Advanced collection overview**

**// 402 namedtuple**

import collections # create a Point namedtuple

Point = collections.namedtuple("Point", "x y")

p1 = Point(10, 20); print(p1.x, p1.y)

p3 = p1.\_replace(x=100) # use \_replace create new instance

print(p1) # Point(x=10, y=20), so p1 remain same, no change

print(p3) # Point(x=100, y=20)

**// 403 defaultdict**

from collections import defaultdict

# define a list of items that we want to count

fruits = ['apple', 'pear', 'orange', 'banana',

'apple', 'grape', 'banana', 'banana']

# use a dictionary to count each element

fruitCounter = **defaultdict(int)**

# Count the elements in the list

for fruit in fruits: **fruitCounter[fruit] += 1**

# print the result

for (k, v) in fruitCounter.items(): print(k + ": " + str(v))

# apple: 2 pear: 1 orange: 1

# banana: 3 grape: 1

fruitCounter = **defaultdict(lambda: 10)** # start each with 10

# apple: 12 pear: 11 orange: 11

# banana: 13 grape: 11

**// 404 Counters**

from collections import Counter

# list of students in class 1

class1 = ["Bob", "James", "Chad", "Darcy", "Penny", "Hannah"

"Kevin", "James", "Melanie", "Becky", "Steve", "Frank"]

# list of students in class 2

class2 = ["Bill", "Barry", "Cindy", "Debbie", "Frank",

"Gabby", "Kelly", "James", "Joe", "Sam", "Tara", "Ziggy"]

# Create a Counter for class1 and class2

c1 = Counter(class1); c2 = Counter(class2)

print(c1["James"]) # How many students class 1 name James?

# How many students are in class 1?

print(sum(c1.values()), "students in class 1")

c1.update(class2) # Combine the two classes

print(sum(c1.values()), "students in class 1 and 2")

# 23 students in class 1 and 2

# What's the most common name in the two classes?

print(most\_common, c1.most\_common(3))

# most\_common [('James', 3), ('Frank', 2), ('Bob', 1)]

mc = c1.most\_common(3);

print('most\_common\_name', mc[0][0]) # James

c1.subtract(class2) # Separate the classes again

print('most\_common', c1.most\_common(3))

print(c1 & c2) # What's common between the two classes?

# Counter({'James': 1, 'Frank': 1})

**// 405 OrderedDict**

**# remember order of insertion to dict**

**from collections import OrderedDict**

# list of sport teams with wins and losses

sportTeams = [("Royals", (18, 12)), ("Rockets", (24, 6)),

("Cardinals", (20, 10)), ("Dragons", (22, 8)),

("Kings", (15, 15)), ("Chargers", (20, 10)),

("Jets", (16, 14)), ("Warriors", (25, 5))]

# sort the teams by number of wins

sortedTeams = sorted(sportTeams, key=lambda t: t[1][0], reverse=True)

print(sortedTeams)

print(sortedTeams[0]) # ('Warriors', (25, 5))

# create an ordered dictionary of the teams

teams = **OrderedDict(sortedTeams)**

print(teams)

# Use popitem to remove the top item

tm, wl = **teams.popitem(False)**

print("Top team: ", tm, wl) # Warriors (25, 5)

# What are next the top 4 teams?

for **i, team in enumerate(teams, start=1):**

print(i, team) # 1 Rockets

if i == 4: break

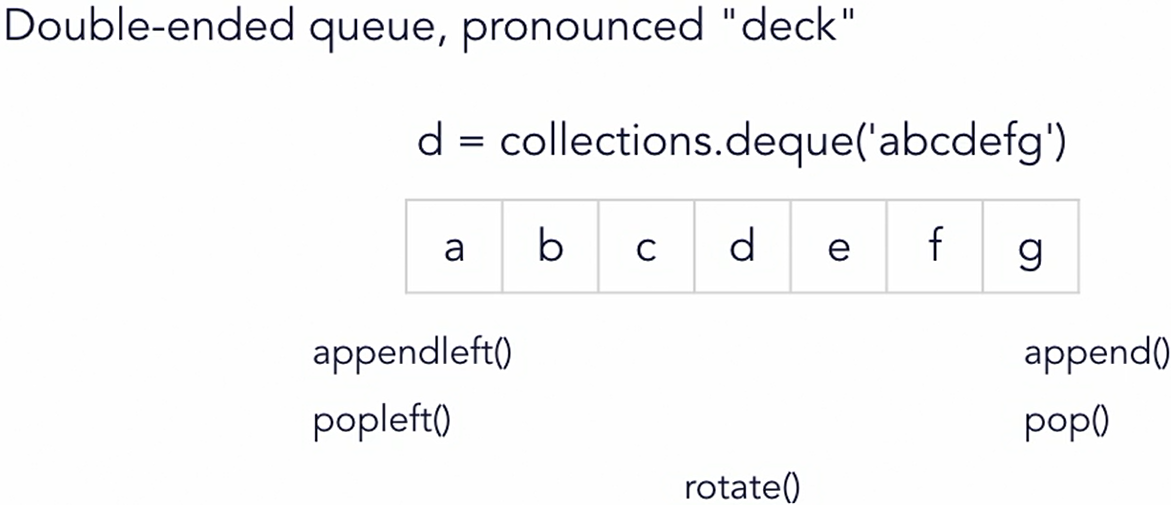
# test for equality

a = OrderedDict({"a": 1, "b": 2, "c": 3})

b = OrderedDict({"a": 1, "c": 3, "b": 2})

print("Equality test: ", **a == b)** # False

**// 406 Deque Objects**

**b/w stack and queue, a hybrid object with memory efficient**

import collections

import string

# initialize a deque with lowercase letters

d = **collections.deque(string.ascii\_lowercase)**

# deques support the len() function

print("Item count: " + **str(len(d))**) # 26

# deques can be iterated over

for elem in d: print(**elem.upper(), end=",")**

# A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z,

# manipulate items from either end

d.**pop()**  # removes last item in deque

d.**popleft()** # removes first item in deque

d.**append(2)** # adds 2 as last item in deque

d.**appendleft(1)** # adds 1 as first item in deque

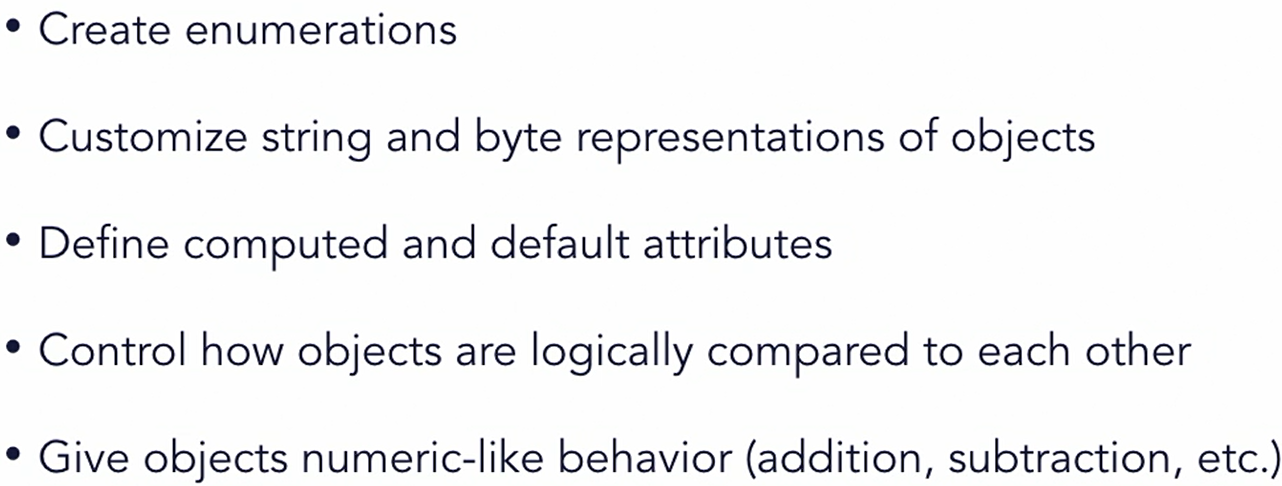
print(d)

# deque([1, 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 2])

d**.rotate(10)** # rotate the deque

print(d)

# deque([**'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 2, 1,** 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p'])

**// 501 Advance classes overview**

**// 502 Enumeration class**

**from enum import Enum, unique, auto**

**@unique** **# it restricts same value for keys**

class Fruit(Enum):

APPLE = 1

BANANA = 2

ORANGE = 3

TOMATO = 4

# POTATO = 1 **# this is valid**

**# two key can have same value**

**# but not when using @unique**

PEAR = auto() # will assign value 5 in this case

# enums have human-readable values and types

print(Fruit.APPLE); # Fruit.APPLE

print(type(Fruit.APPLE)) # <enum ‘Fruit’>

print(repr(Fruit.APPLE)) # <Fruit.APPLE: 1>

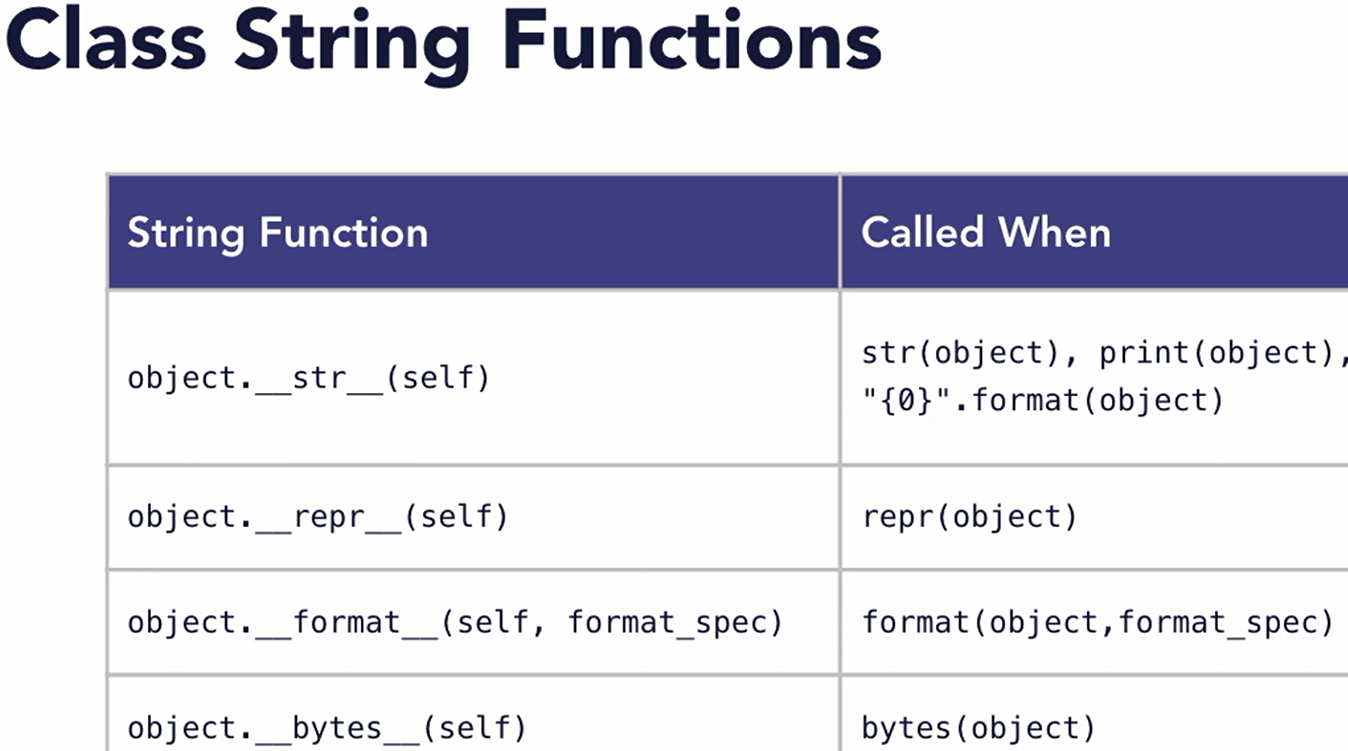
print(Fruit.APPLE.name, Fruit.APPLE.value) # APPLE 1

print(Fruit.PEAR.value) # 5 # print the auto-generated value

# enums are hashable - can be used as keys

myFruits = {}; myFruits[Fruit.BANANA] = "Come Mr. Tally"

print(myFruits[Fruit.BANANA]) # Come Mr. Tally

**// 503 Class string values**

class Person():

def **\_\_init\_\_(self**): self.fname = "Joe"; self.age = 25

# use \_\_repr\_\_ to create a string useful for debugging

def **\_\_repr\_\_(**self):

return "<Person Class - fname:{0},

age{1}>".format(self.fname, self.age)

# use str for a more human-readable string

def **\_\_str\_\_(**self):

return "Person ({0} is {1})".format(self.fname, self.age)

def **\_\_bytes\_\_(**self):

val = "Person:{0}:{1}".format(self.fname, self.age)

return **bytes(val.encode('utf-8'))**

def **\_\_format\_\_(**self, format\_spec): return " - format - "

cls1 = Person()

print(**str(**cls1)) # Person (Joe is 25)

print(**repr(**cls1)) # <Person Class - fname:Joe, age25>

print("Formatted: {0**}".format(cls1)**) # Formatted: - format -

print(**format(**cls1)) # - format -

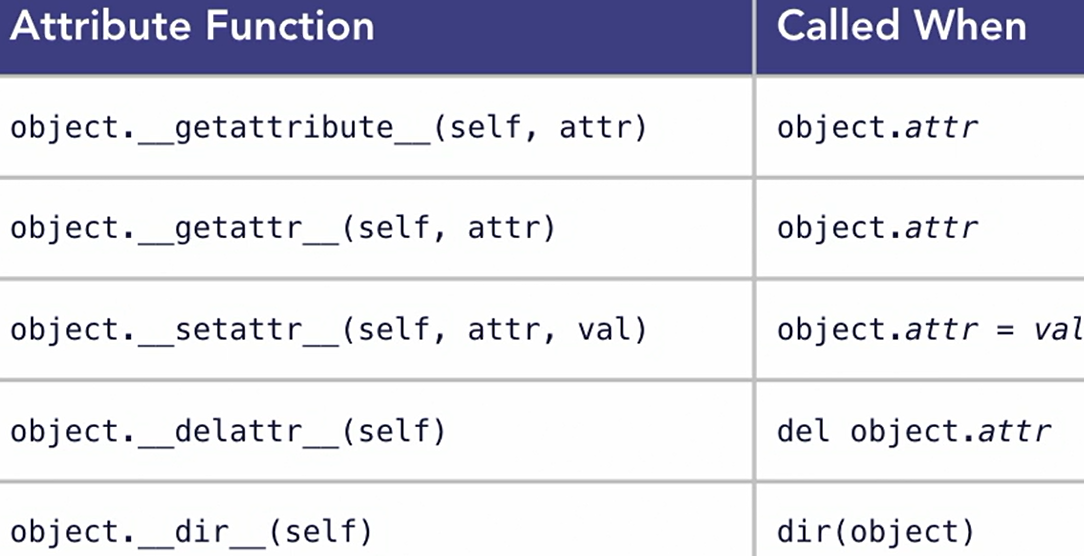
print(**bytes(**cls1)) # b'Person:Joe:25'

**// 504 Computed attributes**

**Py gives opportunity to perform desired process get/set**

**\_\_getattr\_\_ when attr not found on object**

**\_\_getattribute\_\_ called first everytime for attribute get**

**\_\_dir\_\_ when dir(obj) used on object**

class myColor():

def **\_\_init\_\_(**self):

self.red = 50; self.green = 75; self.blue = 100

# use getattr to dynamically return a value

def **\_\_getattr\_\_(self, attr):**

if attr == "rgbcolor": return (self.red, self.green, self.blue)

elif attr == "hexcolor":

return "#{0:02x}{1:02x}{2:02x}"

.format(self.red, self.green, self.blue)

**else: raise AttributeError**

# use setattr to dynamically return a value

def **\_\_setattr\_\_(self, attr, val):**

if attr == "rgbcolor": self.red = val[0];

self.green = val[1]; self.blue = val[2]

else: **super().\_\_setattr\_\_(attr, val)**

# use dir to list the available properties

def **\_\_dir\_\_(self):** return ("rgbcolor", "hexcolor")

**# defining return data type and self argument for function**

def new\_red**(self) -> int**: return self.red # custom function

cls1 = myColor() # create an instance of myColor

# print the value of a computed attribute

print(cls1.rgbcolor) # (50, 75, 100)

print(cls1.hexcolor) # #324b64

# set the value of a computed attribute

cls1.rgbcolor = (125, 200, 86)

print(cls1.rgbcolor) # (125, 200, 86)

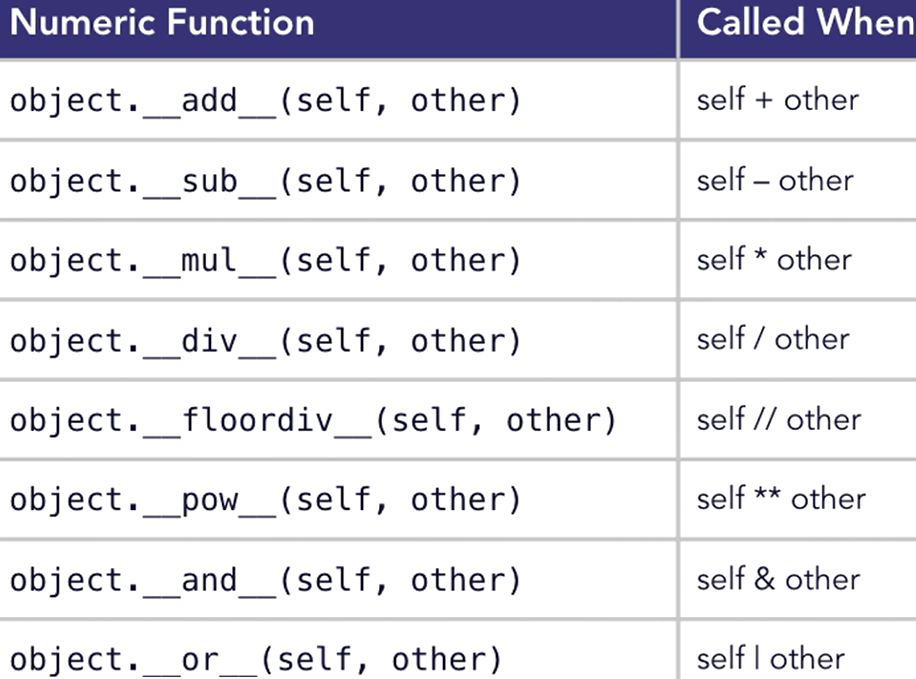
print(cls1.hexcolor) # #7dc856

print(cls1.red) # 125 # access a regular attribute

# list the available attributes

print(**dir(cls1**)) # ['hexcolor', 'rgbcolor']

print(cls1**.new\_red())** # 125

**// 505 Class numeric operators**

# give objects number-like behavior

class Point():

def **\_\_init\_\_(self**, x, y): self.x = x; self.y = y

def **\_\_repr\_\_(**self):

return "<Point x:{0},y:{1}>".format(self.x, self.y)

def **\_\_add\_\_(**self, other): # implement addition

return Point(self.x + other.x, self.y + other.y)

def **\_\_sub\_\_(**self, other): # implement subtraction

return Point(self.x - other.x, self.y - other.y)

def **\_\_iadd\_\_(**self, other): # in-place addition

self.x += other.x; self.y += other.y; return self

# Declare some points

p1 = Point(10, 20); p2 = Point(30, 30)

print(p1, p2) # <Point x:10,y:20> <Point x:30,y:30>

p3 = p1 + p2 # Add two points

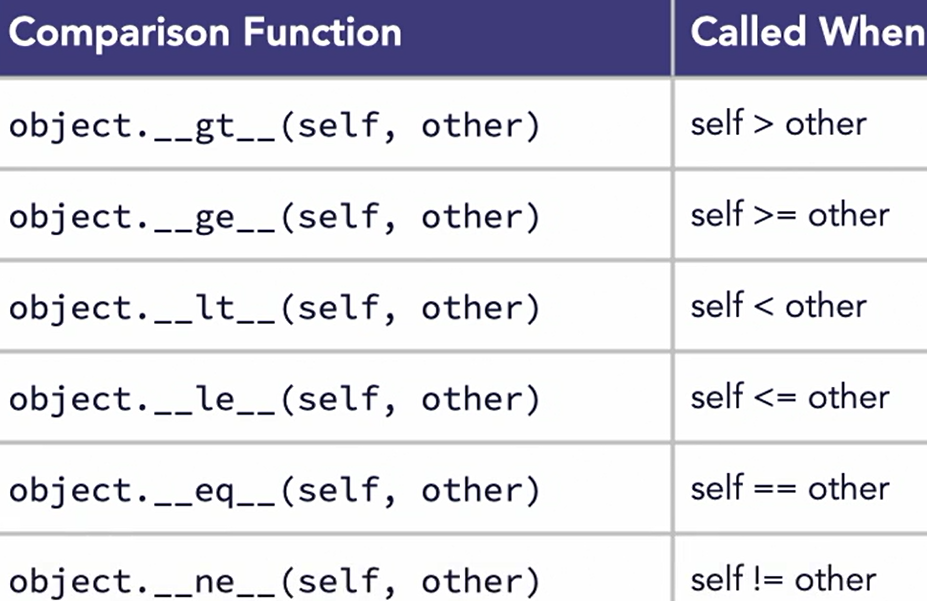
print(p3) # <Point x:40,y:50>

p4 = p2 - p1 # subtract two points

print(p4) # <Point x:20,y:10>

p1 += p2 # Perform in-place addition

print(p1) # <Point x:40,y:50>

**// 506 Object Comparisons**

class Employee():

def \_\_init\_\_(self, level, yrs): self.level = level; self.yrs = yrs

**def \_\_ge\_\_(**self, other):

if self.level == other.level: return self.yrs >= other.yrs

return self.level >= other.level

def **\_\_gt\_\_(**self, other):

if self.level == other.level: return self.yrs > other.yrs

return self.level > other.level

def **\_\_lt\_\_(**self, other):

if self.level == other.level: return self.yrs < other.yrs

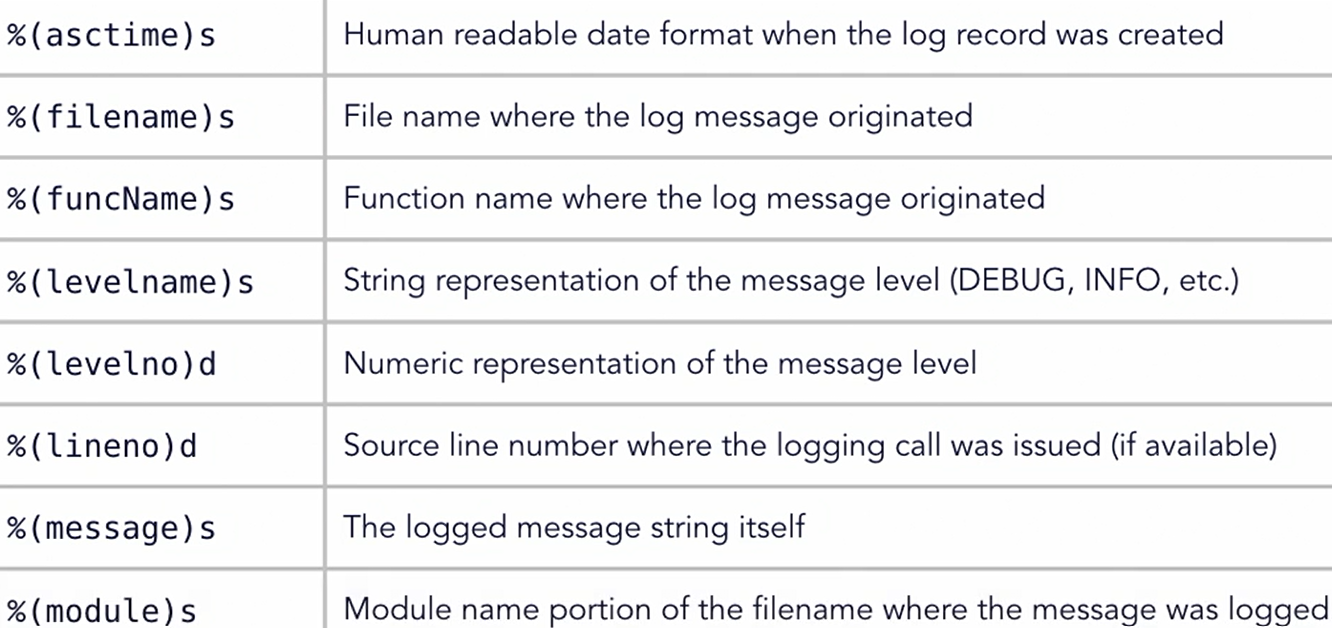
return self.level < other.level

def **\_\_le\_\_(**self, other):

if self.level == other.level: return self.yrs <= other.yrs

return self.level <= other.level

def **\_\_eq\_\_(**self, other): return self.level == other.level

dept = []

dept.append(Employee(5, 9)); dept.append(Employee(4, 12))

dept.append(Employee(6, 6)); dept**.append(Employee(5, 13))**

print(**bool(dept[0] > dept[1]))** # True# Who's more senior?

print(bool(dept[3] < dept[2])) # True

emps = **sorted(dept)** # 4, 5, 5, 6 # sort the items

**// 601 Logging Overview // 602 logging module**

**# use the built-in logging module**

import **logging**

**# Use basicConfig to configure logging**

**# this is only executed once, subsequent calls to**

**# basicConfig will have no effect**

logging.**basicConfig(level=logging.DEBUG,**

**filemode="w",**

**filename="output.log")**

logging.debug("This is a debug-level log message")

logging.info("This is an info-level log message")

logging.warning("This is a warning-level message")

logging.error("This is an error-level message")

logging.critical("This is a critical-level message")

# Output formatted string to the log

logging.info("Here's a {} variable and an int: {}".format("string", 10))

# Above code writes logs to file at root folder location as :

# output.log

DEBUG:root:This is a debug-level log message

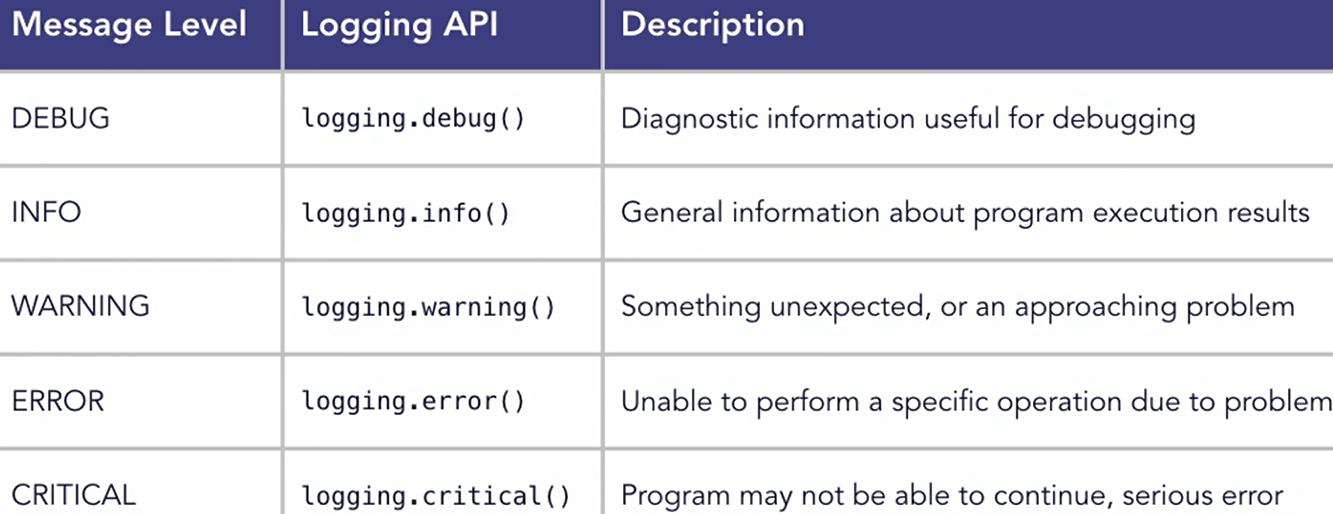
INFO:root:This is an info-level log message

WARNING:root:This is a warning-level message

**ERROR**:root:This is an error-level message

**CRITICAL**:root:This is a critical-level message

INFO:root:Here's a string variable and an int: 10



**// 603 Custom logging**

extData = {'user': 'joem@example.com'}

**# use a custom formatting specification**

fmtStr = **"%(asctime)s: %(levelname)s: %(funcName)s Line:%(lineno)d User:%(user)s %(message)s"**

dateStr = "%m/%d/%Y %I:%M:%S %p"

logging.basicConfig(filename="output.log",

level=logging.DEBUG, **format=fmtStr, datefmt=dateStr**)

logging.info("This is an info-level log message", **extra=extData**)

logging.warning("This is a warning-level message", extra=extData)

# Above code writes logs to file at root folder location as :

# output.log

05/01/2024 05:11:20 PM: INFO: main Line:18 User:joem@example.com This is an info-level log message

**05/01/2024 05:11:20 PM: WARNING: main Line:19 User:joem@example.com** This is a warning-level message

**// 701 Comprehensions**

**Comprehensions are simplification for code using list & map**

Example:

list ( map ( fahrenheit\_to\_celsius, [ 32, 34, 36] )

TO

[ ( t \* 9/5 ) + 32 for t in [ 32, 34, 36 ] ]

**The smaller syntax for loop is comprehension**

It can be used to create list, dictionaries and sets also

**// 702 List comprehensions**

evens = [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]; count = 'even => '

even\_a = list(

map(lambda e: count + str(e), filter(

lambda e: e > 0 and e < 16, evens)))

print(even\_a)

# Derive new list of numbers using comprehensions

even\_b = [count + str(e) for e in evens if e > 0 and e < 16]

print(even\_b) # even\_a == even\_b

**// 703 Dictionary comprehensions**

# define a list of temperature values

ctemps = [0, 12, 34, 100]

# Use a comprehension to build a dictionary

tempDict = {**t: (t \* 9/5) + 32** for t in ctemps if t < 100}

print(tempDict) # {0: 32.0, 12: 53.6, 34: 93.2}

print(tempDict[12]) # 53.6

# Merge two dictionaries with a comprehension

team1 = {"Jones": 24, "Jameson": 18, "Smith": 58, "Burns": 7}

team2 = {"White": 12, "Macke": 88, "Perce": 4}

newTeam = {k: v for **team** in (team1, team2) for k, v in **team.items**()}

print(newTeam)

# {'Jones': 24, 'Jameson': 18, 'Smith': 58, 'Burns': 7, 'White': 12, 'Macke': 88, 'Perce': 4}

**# from py 3.9**

newTeam = **team1 | team2**; print(newTeam)

# {'Jones': 24, 'Jameson': 18, 'Smith': 58, 'Burns': 7, 'White': 12, 'Macke': 88, 'Perce': 4}

newTeam = **{ \*\*team1, \*\*team2 } # from py 3.5**

**// 704 Set comprehensions**

**# Set used to contain unique value in curly braces format**

ctemps = [5, 10, 12, 14, 10, 23, 41, 30, 12, 24, 12, 18, 29]

# build a set of unique Fahrenheit temperatures

ftemps\_set = {(t \* 9/5) + 32 for t in ctemps}

print(ftemps\_set) # {64.4, 73.4, 41.0, 105.8, … }

# build a set from an input source

sTemp = "The quick brown fox jumped over the lazy dog"

chars\_set = {c.upper() for c in sTemp if not c.isspace()}

print(chars\_set) # {'M', 'W', 'T', 'C', 'V', 'E', 'J', 'O', ….. }

miss\_list = [a for a in string.ascii\_uppercase

if a not in chars\_set]

print(miss\_list) # [‘S’]

**Lyn: Python Data Analysis [ Michele Vallisneri ] // skipped**

**// 201 Install Anaconda**

It includes: python, numpy, pandas, matplotlib

Anaconda is an open-source distribution of the Python and R programming languages for data science that aims to simplify package management and deployment. Package versions in Anaconda are managed by the package management system, conda, which analyzes the current environment before executing an installation to avoid disrupting other frameworks and packages

The Anaconda distribution comes with over 250 packages automatically installed

<https://www.anaconda.com/pricing>

**// 103 Working with Jupyter Notebook**

From anaconda navigator

Find Jupyter app on home

Click launch

Create new notebook

Type you code to execute

Press Shift + Enter to execute

To write any comment, from drop-down

Select 🡪 Markdown

copy command as: Esc + c

cut command as: Esc + x

paste command as: Esc + v

delete a cell: d + d

You can use menus as well

**// 105 py in cloud via azure notebooks**

**// 201 Warmup with loops**

Loop needs interables such as:

list, dic, set, interator or generators

import math

import collections

**import numpy as np**

**import pandas as pd**

**import matplotlib.pyplot as pp**

**// 202 Sequence**

List, tuple and slicing syntax

nephews = ["Huey", "Dewey", "Louie"]

nephews**[-1**] // “Louie”

nephews**[-2**] // “Dewey”

**.append, .extend, .insert, .remove, +, .sort, sorted reverse**

nephews.**extend**([‘a’, ‘b’]) // Extend to insert multiple

nephews.insert(0, ‘aa’) // insert ‘aa’ on position 0 index

del nephews[0]

ducks = nephews **+** ['e','f] // add and return

reverse\_ducks = **sorted**(ducks, **reverse=True**)

// reverse means descending order

sq = [1,4,9,16,25,36,49]

sq[**0:2**] // 1, 4 // start at 0 and end at 2

sq**[:4]** // 1,4,9,16 // start at 0 and end at 4

sq**[:]** // copy full list

sq**[0:7:2]** // 1,0,25,49 // skip 2

sq**[-3:-1]** // 25,36 // starts from -3 and ends -1

sq**[2:4] = ['four', 'nine']** // replace using slicing index

print(sq) // [1, 4, 'four', 'nine', 25, 36, 49]

**del sq[4:6]**

print(sq) // [1, 4, 'four', 'nine', 49]

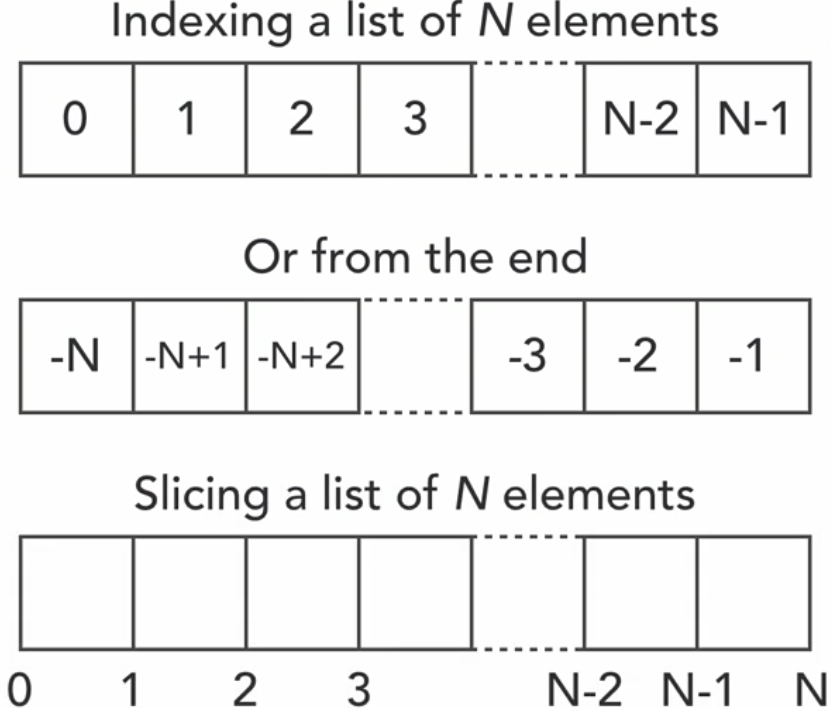
Tuples are immutable collections of variables, so they can't be changed after creation. Sets are mutable collections of variables, so sets can be changed after creation.

| **List** | **Set** | **Tuple** |
| --- | --- | --- |
| Lists is Mutable | Set is Mutable | Tuple is Immutable |
| It is Ordered collection of items | It is Unordered collection of items | It is Ordered collection of items |
| Items in list can be replaced or changed | Items in set cannot be changed or replaced but you can remove and add new items. | Items in tuple cannot be changed or replaced |

**print(set\_a[1]) // TypeError 'set' object is not subscriptable**

**tuple\_a[1] = 0** **// TypeError: 'tuple' object does not support item assignment**

====== ====== ======

**Variable assignation using tuple**

**(a,b) = (1, 2)**

**c, d = 3, 4**

**print(a,b,c,d)** // 1, 2, 3, 4

====== ====== ======

def three\_args(a, b, c): print(a, b, c)

my\_args = (1,2,3); three\_args(\*my\_args)

**// 203 Dictionaries and sets**

caps = { ‘A’: ‘a\_val’, ‘B’: ‘b\_val’, ‘C’: ‘c\_val’ }# Dictionaries

morecaps = { ‘D’: ‘d\_val’, ‘E’: ‘e\_val’ }

print(‘A’ in caps) # True

caps **+** morecaps # TypeError: unsupported

caps.update(morecaps) # add morecaps to caps dict

del caps[‘E’] # delete item from caps dict

for **k, v in caps.items**(): print(k, v)

# A, a\_val B, b\_val C, c\_val ….

caps.values() # dict\_values([‘a\_val’,’b\_val’,’c\_val ….

caps.keys() # dict\_values([‘A’,’B’,’C’, ….

**Note: The keys in dict can be any type**

birthdays = {(7,15): 'Michele', (3,14): 'Albert'}

print(birthdays[(7,15)]) # ‘Michele’

b\_dict = { **hash(3)**: 'abc' }

print(b\_dict[hash(3)]) # abc

**Sets:**

a\_set = {'America', 'Europe', 'Asia', 'Oceania', 'Africa', 'Africa'}

print('Africa' in continents) # True

**Other methods in set:**

**clear, copy, difference, intersection, issubset, issuperset,**

**symmetric\_difference, union, update, discard, …**

a\_set.**add**('Antarctica')

a\_set.**remove**('Antarctica')

**list methods:** [**w3schools**](https://www.w3schools.com/python/python_ref_list.asp)

**set methods:** [**w3schools**](https://www.w3schools.com/python/python_ref_set.asp)

**tuple methods:** [**w3schools**](https://www.w3schools.com/python/python_ref_tuple.asp)

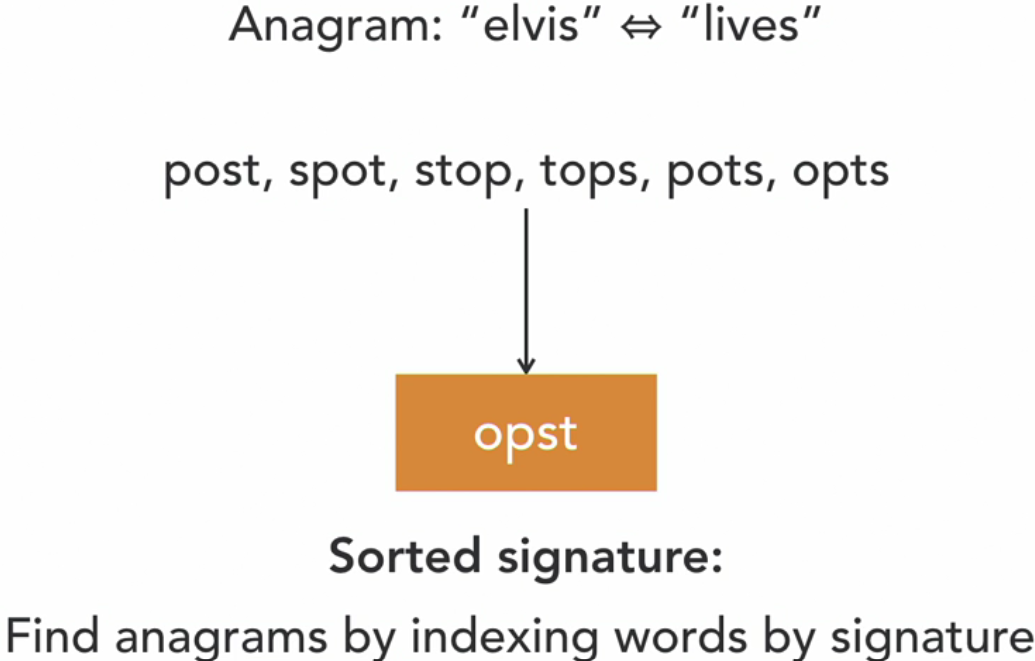
**dict methods:** [**w3schools**](https://www.w3schools.com/python/python_ref_dictionary.asp)

**string methods:** [**w3schools**](https://www.w3schools.com/python/python_ref_string.asp)

**file methods:** [**w3schools**](https://www.w3schools.com/python/python_ref_file.asp)

**python keywords:** [**w3schools**](https://www.w3schools.com/python/python_ref_keywords.asp)

**py exceptions:** [**w3schools**](https://www.w3schools.com/python/python_ref_exceptions.asp)

**// 204 Comprehensions**

[i\*\*2 for i in range(1, 11) if i\*\*2 % 4 == 0]

# [4, 16, 36, 64, 100] // values divisible by 4

sum(i\*\*2 for i in range(1, 11)) # sum squares 1^2 to 10^2

# nested comprehension

counting = [ **j** for **i** in range(1, 11) **for j** in range(1, **i+1**) ]

**// 205 Advance python containers**

**ptype = collections.namedtuple('p'**, ['fn’, 'ln', "b"])

a\_ptype = ptype(‘a’,’b’,’c’)

**type(a\_ptype) # \_\_main\_\_.ptype**

**isinstance(a\_ptype, ptype) # True**

print(a\_ptype) # p(fn=’a’, ln=’b’, b=’c’)

**b\_ptype = ptype(**fn=’ba’, ln=’bb’, b=’bc’)

print(b\_ptype) # p(fn=’ba’, ln=’bb’, b=’bc’ )

print(b\_ptype.ln) # bb

ps = [("a1", "b1", "c1"), ("a2", "b2", "c2"),

("a3", "b3", "c3"), ("a4", "b4", "c4")]

c\_ptype = ptype(\*ps[0])

print(c\_ptype) # p(fn=’a1’, ln=’b1’, b=’c1’ )

d\_ptype = [p(\*i) for i in ps]; print(d\_ptype)

# [p(fn=’a1’, ln=’b1’, b=’c1’ ), p(fn=’a2’, ln=’b2’, b=’c2’ ), p(fn=’a3’, ln=’b3’, b=’c3’ ), p(fn=’a4’, ln=’b4’, b=’c4’ ) ]

====== ====== ======

**def anynamefunc(): return ‘i don know’**

**q = collections.defaultdict(anynamefunc)**

q['The meaning of life']

q['new life']

print(q)

# defaultdict(<function \_\_main\_\_.anynamefunc()>,

{'The meaning of life': ‘i don know’,

'new life': ‘i don know’})

====== ====== ======

**from dataclasses import dataclass**

**@dataclass**

**class pclass**: fn: str; ln: str; b: str = 'unknown'

a\_pclass = pclass(‘a1’,’b1’)

print(a\_plass) # pclass(fn=’a1’,ln=’b1’,b=’unknown’)

**// 301 Anagrams**

Anagrams means finding possible words our of given characters. For Example:

From 2 characters o and n, we can make on and no

**// 202 Loading text from file**

words = sorted({

line.strip().lower() for line in open('words.txt', 'r') })

**// 203 Finding anagrams**

sorted("elvis") == sorted("lives") # True

====== ====== ======

**Lyn: Advanced Python Working with Databases**

**// 201 A**

**// 202 A**

**// 203 A**

**// 204 A**

**// 05 A**

**// 06 A**

**// 201 A**

**// 202 A**

**// 203 A**

**// 204 A**

**// 05 A**

**// 06 A**

**// 201 A**

**// 202 A**

**// 203 A**

**// 204 A**

**// 05 A**

**// 06 A**

**// 201 A**

**// 202 A**

**// 203 A**

**// 204 A**

**// 05 A**

**// 06 A**

**// 201 A**

**// 202 A**

**// 203 A**

**// 204 A**

**// 05 A**

**// 06 A**

**// 201 A**

**// 202 A**

**// 203 A**

**// 204 A**

**// 05 A**

**// 06 A**

**// 201 A**

**// 202 A**

**// 203 A**

**// 204 A**

**// 05 A**

**// 06 A**

**// 201 A**

**// 202 A**

**// 203 A**

**// 204 A**

**// 05 A**

**// 06 A**

**Lyn: Python Design Patterns**

**// 201 A**

**// 202 A**

**// 203 A**

**// 204 A**

**// 05 A**

**// 06 A**

**// 201 A**

**// 202 A**

**// 203 A**

**// 204 A**

**// 05 A**

**// 06 A**

**// 201 A**

**// 202 A**

**// 203 A**

**// 204 A**

**// 05 A**

**// 06 A**

**// 201 A**

**// 202 A**

**// 203 A**

**// 204 A**

**// 05 A**

**// 06 A**

**// 201 A**

**// 202 A**

**// 203 A**

**// 204 A**

**// 05 A**

**// 06 A**

**// 201 A**

**// 202 A**

**// 203 A**

**// 204 A**

**// 05 A**

**// 06 A**

**// 201 A**

**// 202 A**

**// 203 A**

**// 204 A**

**// 05 A**

**// 06 A**

**// 201 A**

**// 202 A**

**// 203 A**

**// 204 A**

**// 05 A**

**// 06 A**

**Lyn: Python Advanced Design Patterns**

**// 201 A**

**// 202 A**

**// 203 A**

**// 204 A**

**// 05 A**

**// 06 A**

**// 201 A**

**// 202 A**

**// 203 A**

**// 204 A**

**// 05 A**

**// 06 A**

**// 201 A**

**// 202 A**

**// 203 A**

**// 204 A**

**// 05 A**

**// 06 A**

**// 201 A**

**// 202 A**

**// 203 A**

**// 204 A**

**// 05 A**

**// 06 A**

**// 201 A**

**// 202 A**

**// 203 A**

**// 204 A**

**// 05 A**

**// 06 A**

**// 201 A**

**// 202 A**

**// 203 A**

**// 204 A**

**// 05 A**

**// 06 A**

**// 201 A**

**// 202 A**

**// 203 A**

**// 204 A**

**// 05 A**

**// 06 A**

**// 201 A**

**// 202 A**

**// 203 A**

**// 204 A**

**// 05 A**

**// 06 A**