Data Types: collections, Lists, Dictionaries, Tuples, Set, Frozen Set
- All variables in python are first class objects. Objects may be mutable or immutable.
- Most fundamental data types in python are immutable: numbers, strings, tuples
  Mutable objects are: lists, dictionaries, set etc.
- A variable in python is just a reference to the object.

Tuples are immutable list, tuples elements cannot be changed.
But, if the element of the tuple itself is mutable data type like list, it's nested items can be changed.***
Use tuple for heterogenous (different) data types and list for homogeneous (similar) data types
If you have data that doesn't change, implementing it as a tuple will guarantee that it will remain write-protected.
Since the tuples are immutable, iterating through tuple is faster than list.
Tuples that contain immutable elements, can be used as key for a dict, with list, this is not possible.

Lists are like array

Set elements are unique & immutable, Set elements are unordered, Set itself is mutable.
We can add or remove items from a set
We can do operations like union, intersection, symmetric diff on sets
Set can have any number of different types (float, integer, tuple etc.), but no mutable elements list, set or dict.

Frozen set is an immutable set, its element cannot be changed, once assigned
Being immutable, it does not have methods that add or remove other elements
Tuples are immutable list, frozen set is immutable set
Sets, being mutable are un-hashable, cannot be used a dict key, frozen set are hashable and can be used as dict key.

 # Data types (no conclusion on id(variable))

```
>>> x=42
>>> id(x)
17214352 <== the value changed
>>> y=42
>>> id(y)
17214352 <== same as before
```

```
>>> x = dict(x=42)
>>> id(x)
17763936
>>> y=dict(x=42)
>>> id (y)
17764224
>>> x is y
True
```

```
>>> a = True
>>> type (a)
<type 'bool'>
>>> id (a)
8970832
>>> id(True)
8970832
```

 # String / Tuples

 # String

```
>>> s='This is a string'
>>> words=s.split()
>>> words
[‘This', 'is', 'a', 'string',
'of', 'words']
>>> new=':'.join(words)
>>> new
'This:is:a:string:of:words'
>>> ', '.join(words)
'This, is, a, string, of, words'
```

s.split()
s.join()
s.find(‘xyz’)
s.replace(‘a’,’A’)
s.strip()
s.rstrip(), s.lstrip()
s.count(‘T’)
s.lower()
~~**s.capitalize()**~~
s.swapcase()
dir(s) display all methods

 # Tuples

```
>>> t1=(1,)
>>> type(t1) # class tuple
>>> t2=(1)
>>> type(t2) # class int
>>> print('a' in t)
True
>>> t1=(1,2,3)
>>> t2=(4,5)
>>> t1+t2
(1,2,3,4,5)
>>> t2*2 O/P:(4,5,4,5)
```

t.count(‘x’)
t.index(‘x’)
enumerate(t)
len(t)
max(t) || min(t)
sorted(t) # O/P: a list
sum(t)
del(t)
any(t)
Returns true if any element is true

 # Lists / Sets

 # Lists

```
l=[2,1,3]
l.sort()
l   O/P: [1, 2, 3]
l.sort(reverse=True)
l   O/P: [3, 2, 1]

l1=[4,7,5]
sorted(l1) O/P:[4, 5, 7]

l.extend(l1)
O/P: [3, 2, 1, 4, 7, 5]

li=[[2,6],[1,3],[5,4]]
li.sort(key=lambda x:
x[0])
li.sort(key=lambda x:
x[1])
O/P: [[1, 3], [2, 6],
[5, 4]]
O/P: [[1, 3], [5, 4],
[2, 6]]
```

 # Same as Tuples + these..

l.append()
l.extend()
l.insert(0,11)
l.pop(1)
l.pop()
l.remove(2)

 # Sets

```
>>> myset = {1,2,3,4,5}
>>> myset=set("apple")
>>> print(myset) O/P: {'l', 'e', 'a',
'p'}
>>> myset=set(["apple"])
>>> print(myset) O/P: {'apple'}
>>> A={1,2,3,4,5}
>>> B={4,5,6,7,8}
>>> print(A|B)   # A.union(B)
{1, 2, 3, 4, 5, 6, 7, 8}
>>> print(A&B)   O/P: {4, 5}
>>> print(A - B) O/P: {1, 2, 3}
>>> print(A ^ B) O/P: {1, 2, 3, 6, 7, 8}
>>> print(A.difference(B))
{1, 2, 3}
>>>
>>> A=frozenset([1,2,3,4])
```

any(), **enumerate()**, **len()**, **max()**, **sorted()**, **sum()**, **add()**, add an element to set
update(), updates like dict
clear(), removes all element
pop(), removes arbitary
discard(), removes element if it is a member
copy() #return shallow copy
difference(), **intersection()**, **union()**
symmetric_difference()
issubset(), **isdisjoint()**

Slicing / Dict

```
# Slicing
x=['first','second','third','fourth']
x[1:-1] O/P: ['second', 'third']

x[0:3]  O/P: ['first', 'second',
'third']

x[-2:-1] O/P: ['third']
x[:3] O/P: ['first', 'second',
           'third']
x[2:] O/P: ['third', 'fourth']
x[:]  O/P: ['first', 'second',
           'third', 'fourth']


l1 = [1,2,3,4,5]
l1[::-1]   O/P: [5, 4, 3, 2, 1]
l1[::-2]   O/P: [5, 3, 1]

l1[::2]    O/P: [1, 3, 5]
l1[3:0:-1] O/P: [4, 3, 2]
l1[3:0:-2] O/P: [4, 2]

list(range(0,5))    O/P; [0, 1, 2, 3, 4]
list(range(2,10,2)) O/P: [2, 4, 6, 8]

list(range(5,0,-1)) O/P: [5, 4, 3, 2, 1]
list(range(5,10,-1))O/P: []

li=[[2,6],[1,3],[5,4]]
li.sort(key=lambda x: x[0])
li.sort(key=lambda x: x[1])
O/P: [[1, 3], [2, 6], [5, 4]]
O/P: [[1, 3], [5, 4], [2, 6]]
```

```
# Dict
choices = dict (
        one = 'first',
        two = 'second',
        three = 'third',
        four = 'fourth'
    )

myDict2={"one":"uno","two":"dos","three":
"tres"}

for k in myDict2: print(k)

myDict2.pop('one') O/P: 'uno'


myDict2.popitem()  O/P: ('three', 'tres')

myDict2.clear()  O/P: {}

myDict2.keys() O/P: dict_keys(['one',
'two'])

myDict2.update({'a':1,'b':2})

O/P: {'one': 1, 'two': 2, 'name': 'eddy',
'a': 1, 'b': 2}

print(sorted(myDict2))
O/P: ['a', 'b', 'name', 'one', 'two']

print(sorted(myDict2, key=len))
O/P: ['a', 'b', 'one', 'two', 'name']
```

```
d.pop('x')  # returns d['x']
d.pop('x', 'Not')
d.popitem() # returns k,v
d.clear()   # clears the dict
d.keys()
d.values()
d.items()
d.get()
d.setdeault('age', 30)
# if d['age'] absent then add
d['age']= None
d.update(d1) # updates d with
common with d1 plus new k,v
```

collections

```
# from collections import Counter
list = [1,2,2,1,2,2,1]
cnt = Counter(list) <<<
print(cnt)   O/P: Counter({2:4, 1:3})
print(cnt[1])O/P: 3

for i in (cnt.elements()): print(i)
O/P: [1,1,1,2,2,2,2]

print(cnt.most_common())  <<<
O/P: [(2,4),(1,3),(3,2)..]

cnt = Counter({1:3, 2:4}) <<<
deduct = {1:1, 2:2}
print(cnt.dsubtract(deduct))
O/P: Counter({1:2, 2:2})


# from collections import defaultdict
# does not throw KeyError
nums = defaultdict(int)
nums['one'] = 1
nums['two'] = 2
print(nums['three']) O/P: 0


# find freq of identical names
count = defaultdict(int)
names = 'Mike John Mike Lou Lou'
for name in names_list.split():
  count[name] += 1
print(count)

O/P: {'Mike': 2, 'John': 1, 'Lou': 2}
```

```
# from collections import OrderedDict
# keys maintains order in the way it
is inserted

od = OrderedDict()
od['a'] = 1
od['b'] = 2
od['c'] = 3

for k,v in od.items(): print(k,v)
a 1
b 2
c 3

# most freq letter will be insert
first
list = ['a', 'a', 'c', 'a', 'c', 'd']
cnt = Counter(list)
od = OrderedDict(cnt.most_common())
for k,v in od.items(): print(k,v)
O/P:
a 3
c 2
d 1

# from collections import deque
# optimized for insert/remove
list = ['a', 'b', 'c']
deq = deque(list)
deq.append('d')
deq.appendleft('e')
print(deq) O/P: e,a,b,c,d

deq.pop()
deq.popleft()
print(deq) O/P: a,b,c

print(deq.count('a')) O/P: 1
```

```
# from collections import ChainMap
# combines dict & mappings, returns
list

d1 = {'a': 1, 'b': 2}
d2 = {'c': 3, 'd': 4}
cm = ChainMap(d1,d2)
print(cm)
O/P: ({'a': 1, 'b': 2}, {'c': 3,
'd': 4})

print(cm('a')) O/P: 1
d2['c'] = 5
print(cm.maps())
O/P: ({'a': 1, 'b': 2}, {'c': 5,
'd': 4})

for k,v in cm: print(k,v)
O/P: # prints k v in each line

# chain maps only takes first b=2
d1 = {'a': 1, 'b': 2}
d2 = {'c': 3, 'b': 4}
print(list(chain_map.keys())
O/P: 'b','a','c'
print(list(chain_map.values())
O/P: 2, 1, 3

d3 = {'e': 5}
new = chain_map.new_child(d3)
print(new)
O/P: ({'a': 1, 'b': 2}, {'c': 5,
'd': 4} {'e': 5})

# from collections import namedtuple
Student = namedtuple('Student',
'fname, lname, age')
s1 = Student('John', 'Clark', '13')
print(s1.fname)  O/P: John
```

Date & Time:

%a | %A = Sun, Mon, ..|Sunday, Monday ..	%p | %P = AM, PM
%b | %B = Jan, Feb, ..|January …	%f = microsec
%d = day of the month	%H | %I = 24 hour format | 12 Hour format
%m = month in 2 digit	%M | %S = u know it
%Y = Year in 4 digit	2021-08-05 15:25:56:792554 = **"%Y-%m-%d %H:%M:%S:%f"**

```
import time

print(str(time.ctime(time.time())))
O/P: Wed Jan 10 19:31:54 2024

time.time() O/P: 1765236943.0288675
```

```
import time

time.localtime()
O/P: time.struct_time(tm_year=2025, tm_mon=12, tm_mday=8, tm_hour=18,
tm_min=35, tm_sec=13, tm_wday=0, tm_yday=342, tm_isdst=0)

time.gmtime()
O/P: time.struct_time(tm_year=2024, tm_mon=1, tm_mday=11, tm_hour=0,
tm_min=32, tm_sec=56, tm_wday=3, tm_yday=11, tm_isdst=0)
```

```
import time

gmt = time.gmtime()
time.strftime("The date is: %Y-%m-%d", gmt)
O/P: 'The date is: 2024-01-11'

time.strftime("The date is: %b %d, %Y", gmt)
O/P: 'The date is: Jan 11, 2024'

time.strftime("It is now: %I %M%p", gmt)
O/P: 'It is now: 12 32AM'
```

```
import time

>>> time.strptime("Saturday, March 8, 2014", "%A, %B %d, %Y")
time.struct_time(tm_year=2014, tm_mon=3, tm_mday=8, tm_hour=0,
tm_min=0, tm_sec=0, tm_wday=5, tm_yday=67, tm_isdst=-1)

>>> dt=time.strptime("Saturday, March 8, 2014", "%A, %B %d, %Y")

>>> time.mktime(dt) O/P: 1394254800.0  ## Epoch time

>>> time.gmtime(1394254800.0)
time.struct_time(tm_year=2024, tm_mon=1, tm_mday=26, tm_hour=2,
tm_min=22, tm_sec=13, tm_wday=4, tm_yday=26, tm_isdst=0)
```

```
# Get current date

from datetime import time

# Input format: time(hour, minute, second)
# time(hour, minute, second, microsecond)

b = time(11, 34, 56)
print("b =", b)   O/P: b = 11:34:56

c = time(hour = 11, minute = 34, second = 56)
print("c =", c)   O/P: c = 11:34:56

d = time(11, 34, 56, 234566)
print("d =", d)   O/P: d = 11:34:56.234566

# Get date from epoch timestamp

from datetime import date

timestamp = date.fromtimestamp(1326244364)
print("Date =", timestamp)
O/P: Date = 2012-01-11
```

```
# We can only import date class from the datetime module.

from datetime import date

a = date(2019, 4, 13)
print(a)  O/P: 2019-04-13

# same as
# d = datetime.date(2019, 4, 13)
# print(d)


# Print today's year, month and day

from datetime import date

# date object of today's date
today = date.today()

print("Current year:", today.year)    O/P: Current year: 2024
print("Current month:", today.month)  O/P: Current month: 1
print("Current day:", today.day)      O/P: Current day: 15
print(today)                          O/P: 2018-12-19
```

```
# Get current date and time

import datetime import datetime

datetime_object = datetime.now()
print(datetime_object)
2018-12-19 09:26:03.478039

# Print year, month, hour, minute, timestamp

from datetime import datetime

a = datetime(2017, 11, 28, 23, 55, 59, 342380)
print("year =", a.year)      O/P: year = 2017
print("month =", a.month)    O/P: month = 11
print("hour =", a.hour)      O/P: hour = 23
print("minute =", a.minute)  O/P: minute = 55
print("timestamp =", a.timestamp())
O/P: timestamp = 1511913359.34238
```

```
# Get date from a timestamp

from datetime import datetime

#datetime(year, month, day)
a = datetime(2018, 11, 28)
print(a)             O/P:  2018-11-28 00:00:00


# datetime(year, month, day, hour, minute, second, microsecond)
b = datetime(2017, 11, 28, 23, 55, 59, 342380)
print(b)             O/P: 2017-11-28 23:55:59.342380

print(b.timestamp()) O/P: 1511913359.34238  # epoch.microsec
```

```
# strftime() & strptime()

from datetime import datetime

# current date and time
now = datetime.now()

t = now.strftime("%H:%M:%S")
print("time:", t)   O/P: time: 04:34:52

s1 = now.strftime("%m/%d/%Y, %H:%M:%S")
# mm/dd/YY H:M:S format
print("s1:", s1)    O/P: s1: 12/26/2018, 04:34:52

s2 = now.strftime("%d/%m/%Y, %H:%M:%S")
# dd/mm/YY H:M:S format
print("s2:", s2)  O/P: s2: 26/12/2018, 04:34:52


from datetime import datetime

date_string = "21 June, 2018"
print("date_string =", date_string) O/P: date_string = 21
June, 2018

date_object = datetime.strptime(date_string, "%d %B, %Y")
print("date_object =", date_object)  O/P: date_object =
2018-06-21 00:00:00
```

```
# Handling timezone

from datetime import datetime

import pytz

local = datetime.now()
print("Local:", local.strftime("%m/%d/%Y, %H:%M:%S"))

tz_NY = pytz.timezone('America/New_York')
datetime_NY = datetime.now(tz_NY)
print("NY:", datetime_NY.strftime("%m/%d/%Y,
%H:%M:%S"))

tz_London = pytz.timezone('Europe/London')
datetime_London = datetime.now(tz_London)
print("London:", datetime_London.strftime("%m/%d/%Y,
%H:%M:%S"))

O/P:
Local time: 2018-12-20 13:10:44.260462
America/New_York time: 2018-12-20 13:10:44.260462
Europe/London time: 2018-12-20 13:10:44.260462
```

```
# Print difference between two timedelta objects

from datetime import timedelta

t1 = timedelta(weeks = 2, days = 5, hours = 1, seconds = 33)
t2 = timedelta(days = 4, hours = 11, minutes = 4, seconds = 54)
t3 = t1 - t2

print("t3 =", t3)
O/P: t3 = 14 days, 13:55:39

# Printing negative time delta
from datetime import timedelta

t1 = timedelta(seconds = 33)
t2 = timedelta(seconds = 54)
t3 = t1 - t2

print("t3 =", t3)
print("t3 =", abs(t3))

O/P:
t3 = -1 day, 23:59:39
t3 = 0:00:21
```

```
# Print difference between two dates and time

from datetime import datetime, date

t1 = date(year = 2018, month = 7, day = 12)
t2 = date(year = 2017, month = 12, day = 23)
t3 = t1 - t2
print("t3 =", t3)

t4 = datetime(year = 2018, month = 7, day = 12,
hour = 7, minute = 9, second = 33)

t5 = datetime(year = 2019, month = 6, day = 10,
hour = 5, minute = 55, second = 13)

t6 = t4 - t5
print("t6 =", t6)

print("type of t3 =", type(t3))
print("type of t6 =", type(t6))

O/P:
t3 = 201 days, 0:00:00
t6 = -333 days, 1:14:20
type of t3 = <class 'datetime.timedelta'>
type of t6 = <class 'datetime.timedelta'>
```

```
from datetime import datetime, timedelta
from pytz import timezone

def convert_timezone(date1, z1, z2):
    '''
Converts time from zone z1 to zone z2
Input format: 02/13/2014 22:39:51:463914
    '''

    if z1 == "EST" and z2 == "EST":
        dateobj = datetime.strptime(date1, "%m/%d/%Y
%H:%M:%S:%f")
        date2 = datetime.strftime(dateobj, "%m/%d/%Y %H:%M")
    else:
        eastern_tz = timezone('US/Eastern')
        # convert z2 to EST
        if z1 == "EST" and z2 == "Singapore":
            other_tz = timezone('Asia/Singapore')
        if z1 == "EST" and z2 == "GB":
            other_tz = timezone('Europe/London')

        t_temp = datetime.strptime(date1,"%m/%d/%Y
%H:%M:%S:%f")
        other_tz_local = other_tz.localize(t_temp)
        t_eastern_local =
other_tz_local.astimezone(eastern_tz)
        date2 = t_eastern_local ## convert timezone to EST

        return date2
```

```
from datetime import datetime, timedelta
from pytz import timezone

def find_time_diff(d, d2):
    '''
    Calculates Time diff up to one decimal round
    '''
    d1_obj = datetime.strptime(d1, "%m/%d/%Y
%H:%M:%S:%f")
    d2_obj = datetime.strptime(d2, "%m/%d/%Y
%H:%M:%S:%f")
    d_diff = (d2_obj - d1_obj)

    return (round(d_diff.total_seconds()/(60*60),1))
```

Ref:
https://ioflood.com/blog/python-timedelta/#:~:text=Python%27s%20timedelta%20is%20a%20function,between%20two%20dates%2C%20and%20more.&text=In%20this%20code%2C%20we%27re,the%20current%20date%20using%20datetime.
https://www.geeksforgeeks.org/calculate-time-difference-in-python/

 # Misc tricks enumerate / *args / **kwargs /exceptions / os, pwd, sys, glob, shutil

```
# 1 enumerate
t = (1, 2, 3, 1, 2, 3)
for i,n in enumerate(t): print(i,n)

# 2 zip
l1 = [2, 3]
l2 = ['a', 'b', 'c']
for k in zip(l1,l2): print(k)

# 3 list and dict comprehension
l10=[i+2 for i in l if i%2 == 0]
d10={i:i+2 for i in l if i%2 == 0}

# 4 *args max=max_num(2,3,18) O/P: 18
def max_num(*num):
   max = num[0]
   for i in num[1:]:
     if i > max:
       max = i
   return(max)

# 5 **kwargs example_kwargs(3,4,x=7,y=20) O/P: 3, 4
27
def example_kwargs(foo=5, bar=10, **kwargs):
   print("value of foo={0} bar={1}".format(foo,bar))
   sum = 0
   for k,v in kwargs.items():
     sum = sum + kwargs[k]

   print("value of kwargs {0}".format(sum))

# common exceptions
ValueError || IOError || TypeError
KeyError || StopIterations
TypeError is when sting is attached in an integer
object e.g. os.environ['HOME']=42
KeyError is when you ref dict value for a non
existing key
You can also do raise ValueError (see code later)
```

```
# os, pwd, sys, glob, shutil
os.getuid() || os.getlogin()
os.getcwd() || os.chdir('..')
os.mkdir('junk1') | os.mkdirs('junk10/jink20')

# returns os.stat_result struct, [0]=perm, [6]=size
os.stat('filename')

os.path.split('/x/y.txt') || os.path.splitest('y.txt')
os.path.join(d,f)
os.path.abspath('file1') || os.path.realpath('file1')
os.environ['baba'] = 'black sheep'

os.walk()

os.listdir('dir1')  # lists all files and dirs
os.remove('file')   # can remove only file

BASEDIR = os.path.dirname(os.path.abspath(__file__))
CONFDIR = os.path.join(BASEDIR,'conf')

# pwd.getpwall() returns pwd.struct_passwd
pwd.getpwuid(os.getuid()) || pwd.getpwall()

sys.exit("bye bye") || sys.stdin() || sys.argv[0|1]
sys.path | sys.version

# returns a list of file and dirs
glob.glob('*')
glob.glob('*.*')     # all files with extension
glob.glob('ab?.*')
glob.glob('*.??')    # e.g. example.py

shutil.copytree('dir1','dir2')
shutil.rmtree('dir1')
shutil.move('file1','subdir')
shutil.copytree('dir1','dir2',ignore=ignore_pattern('*.pyc'))
# ignore_pyc can be a custom function with list of .pyc files
shutil.copytree('dir1','dir2',ignore=ignore_pyc)
```

Idiomatic python (Bad || Good):

```
if foo == True: || if foo:
if raise_shields() == True: || if raise_shields():

if name == 'Tom' or name == 'Dick' or name == "Harry':
  is_generic = True
|| is_generic = name in ('Tom', 'Dick', 'Harry')

print('{} {}'.format(index,element))
|| print('{0} {1}'.format(index,element))


my_list = ['larry','moe',joe']
for element in my_list:
  print(element)


# use else to execute code after for loop

for user in get_all_users():
  if email_is_malformed(email_addr):
    break
else:
  print('all email is valid')

# avoid using '', [] and {} as default params to
functions. Watch below
def f(a,l=[]):
  l.append(a)
  return l

print(f(1)) O/P: [1]
print(f(2)) O/P: [1, 2]
# if you do not want the default value to be shared
# between subsequent calls then do this
def f(a, L=None):
  if L is None:
    L = []
  L.append(a)
  return L

print(f(1)) O/P: [1]
print(f(2)) O/P: [2]

# use ''.join when creating a single string for list
elements
some_list = ['a', 'b', 'c', 'd', 'e']
result_string = ','.join(some_list)

# Chain the string functions instead of doing in steps
formatted_info =
book_info.strip().upper().replace(':",' by')

# Use tuples to unpack data
list_of_values = ['dog', 'fido', 10]

(animal, name, age) = list_of_values
output = ('{name} the {animal} is
    {age}'.format(animal=animal, name=name, age=age))

# Use _ as a placeholder for data in tuple that should
be ignored
(name, age, _, _) = ger_user_info(user)
if age > 21:
  ...

# Avoid using temp variable when performing swap
foo, bar = 'Foo', 'Bar'
(foo, bar) = (bar, foo)

# Use Capital letters when declaring global variables
SECONDS_IN_A_DAY = 60 * 60 * 24

# Use Camel case for Class, variables in lower case
class StringManipulator()
joined_by_underscore = True
```

```
# Use *args and **kwargs to accept arbitrary arguments
< See the example in Misc trics >

# Use List comprehension or Dict comprehension if they fit
user_email =
   {user.name: user.email for user in users_list
                          if user.email}

# Use set comprehension to generate sets concisely
# note, with one key, s = {k ..}  it becomes a set,
# with s = {k:v ..} it becomes a dict
users_first_names = {user.first_name for user in users}

# Use sets to eliminate duplicate entries
unique_surnames = set(employee_surnames)
display(unique_surnames)

# Use set operations as much as possible
A|B = Union, A & B = Intersection, A-B = Difference
A ^ B = Symmetric difference
..
return set(get_active_user_list()) &
       set(get popular_user_list())

# Use generator to lazily load infinite sequences
..

# Prefer generator expression over list comprehension
for item in (name.upper() for name in get_all_users()):
  process_username(item)



# Use context manager when opening a file
with open(path_to_fike,'r') as fh:
  ..

# Use sys.exit(main()) || sys.exit(1) || sys.exit('Yoy…")


# Use * operator to represent the rest of a list
some_list = ['a', 'b', 'c', 'd', 'e']

(first, second, *rest) = some_list
print(rest)
(first, *middle, rest) = some_list
print(middle)
(*head, second, rest) = some_list
print(head)


# Use dict.get() to provide default values
log_severity = config.get('severity', 'Info')


# always use format function to format strings
output = 'Name: {user.name}, Age:
         {user.age}'.format(user=user)

"{0} is the {1}".format("Ambrosia","food")
O/P: 'Ambrosia is the food'

"{food} is the {god}".format(food="Ambrosia",god="food")
O/P: 'Ambrosia is the food'

"{0:10} is food of gods".format("Ambrosia")
O/P: 'Ambrosia   is food of gods'

"{0:>10} is food of gods".format("Ambrosia")
O/P: '  Ambrosia is food of gods'

num_dict = {'e': 2.718, 'pi': 3.14159}
print("%(pi).2f - %(pi).4f - %(e).2f" %num_dict)
O/P: 3.14 - 3.1416 - 2.72. # rounded
```

Lambda/Map/Filter/Zip/Reduce

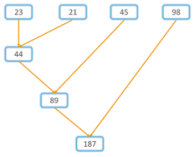
```
# lambda x,y,z, ...: function of(x,y,z,...)

double = lambda x: x*2
print("double: ", double(4))   O/P: 8

# example
y = lambda m,x,c: m*x + c
print("y=mx+b: ", y(1,2,3))

# example. No '' needed
convert_temp =
  {'f2k': lambda deg_f: 273.15 + (deg_f - 32) * 5/9,
   'c2k': lambda deg_c: 273.15 + deg_c }

print("f2k: ", convert_temp['f2k'](32))   f2k:  273.15
print("c2k: ", convert_temp['c2k'](50))   c2k:  323.15
```

```
# map(func, *iterable), it transforms lije map in perl. map can take 3 arguments also

l1 = ['a1', 'b1', 'c1']
upper1 = []

for i in l1:
    upper1.append(i.upper())
print("upper1: ", upper1)


upper2 = list(map(lambda i: i.upper(), l1))
print("upper2: ", upper2)

# example: zip, implement zip with map

l1 = ['a', 'b', 'c', 'd', 'e', 'f']
n1 = [1, 2, 3, 4]

res1 = list(zip(l1,n1))
print("zip result: ", res1)

res2 = list(map(lambda x, y: (x,y), l1, n1))
print("zip using map: ", res2)
```

```
# example

tup = (5, 7, 22, 97, 54, 62, 77, 23)
newtuple = tuple(map(lambda x: x+3 , tup))
print(newtuple)

O/P: (8, 10, 25, 100, 57, 65, 80, 26)
```

```
# I have a list of circle areas with 5 decimal precision. Round each element in the list up to its decimal position places. 1st element -> decimal 1, 2nd -> decimal 2 ..

ca = [ 3.56773, 5.57668, 4.00914, 5.77213, 6.11932]
result = list(map(round, ca, range(1,7)))

print("circle areas: ", result)

O/P: circle areas:  [3.6, 5.58, 4.009, 5.7721, 6.11932]
# explanation:1st elm: round(3.56773,1) -> 2nd elm: ound(5.57668,2)
```

```
# reduce(func, iterable[, initial])
# Find the sum of [23,21,45,98] using reduce

from functools import reduce
reduce(lambda a,b: a+b,[23,21,45,98])
O/P: 187
```




```
# same problem using custom function

from functools import reduce

numbers = [3,4,6,9,34,12]
def custom_sum(first, second):
    return first+second

result = reduce(custom_sum, numbers)
print("Result is: ", result)

Result is:  68
```

```
# example: flatten the list below

l1=[[1, 3, 5], [7, 9], [11, 13, 15]]
from functools import reduce
reduce(list.__add__, [[1, 3, 5], [7, 9], [11, 13, 15]], [])
O/P: [1, 3, 5, 7, 9, 11, 13, 15]

Note: list.__add__([1,2],[3,4]) O/P: [1,2,3,4]
```

```
 # example: find common elements in the list of lists below
num = [[5, 7, 8, 10, 3], [5, 12, 45, 8, 9], [8, 39, 90, 5, 12]]
res = reduce(set.intersection, map(set, num))
print(res)

O/P:  {8, 5}
# map object is iterable, map(set, num) returns 3 sets
```

```
# return Palindrome words from a list of words

words = ['demigod', 'rewire', 'madam']
p_words = list(filter(lambda word: word == word[::-1], words))

print("p_words: ", p_words)

O/P: p_words:  []
```

```
import re (BAD!!, skip)

row = """172.16.0.3 - - [25/Sep/2002:14:04:19 +0200] "GET / HTTP/1.1" 401 - "" "Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.1) Gecko/20020827" """

print(list(map(''.join,
re.findall(r'\"(.*?)\"|\[(.*?)\]|(\S+)', row))))

#['172.16.0.3', '-', '-', '25/Sep/2002:14:04:19 +0200', 'GET / HTTP/1.1', '401', '-', '', 'Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.1) Gecko/20020827']
```

Reading/Writing from various configuration files

```
# databaseconfig.py
#!/usr/bin/env python
import preprocessing

mysql = {
    "host": "localhost",
    "user": "root",
    "passwd": "my secret password",
    "db": "write-math",
}
preprocessing_queue = [
    preprocessing.scale_and_center,
    preprocessing.dot_reduction,
    preprocessing.connect_lines,
]
use_anonymous = True

# JSON data file, config.json
{
    "mysql":{
        "host":"localhost",
        "user":"root",
        "passwd":"my secret password",
        "db":"write-math"
    },
    "other":{
        "preprocessing_queue":[
            "preprocessing.scale_and_center",
            "preprocessing.dot_reduction",
            "preprocessing.connect_lines"
            ],
        "use_anonymous":true
    }
}
```

```
# Reading the file
import json

with open("config.json") as json_data_file:
    data = json.load(json_data_file)
print(data)

# Writing the file
import json
with open("config.json", "w") as outfile:
    json.dump(data, outfile)

# convert a dict to json:
d = {'one': 'uno', 'two': 'dos'}
import json
data = json.dumps(d)


# Reading gzipp'ed log file
import gzip

if __name__ == '__main__':
    with gzip.open('us-log1.log.gz') as fh:
        for line in fh:
            print(line)
```

```
# YAML data file, config.yml
mysql:
    host: localhost
    user: root
    passwd: my secret password
    db: write-math
other:
    preprocessing_queue:
        - preprocessing.scale_and_center
        - preprocessing.dot_reduction
        - preprocessing.connect_lines
    use_anonymous: yes


# config1.yml
document: 1
name: 'erik'
---
document: 2
name: 'config'

..Has to read file with above contents..
..(previous lines stripped)

docs = yaml.safe_load_all(file)
for doc in docs:
        print(doc)
O/P:
{'document': 1, 'name': 'erik'}
{'document': 2, 'name': 'config'}
```

```
# YAML data file, config.yml
# Reading the file
import yaml

with open("config.yml", "r") as ymlfile:
    cfg = yaml.safe_load(ymlfile)

for section in cfg:
   print(section)
   print(cfg["mysql"])
   print(cfg["other"])

O/P:
other
mysql
{
    "passwd": "my secret password",
    "host": "localhost",
    "db": "write-math",
    "user": "root",
}
{
    "preprocessing_queue": [
        "preprocessing.scale_and_center",
        "preprocessing.dot_reduction",
        "preprocessing.connect_lines",
    ],
    "use_anonymous": True,
}
## Writing to a file in yml format
with open("config.yml", "r") as ymlfile:
    data = yaml.safe_load(ymlfile)

with open('config.out.yml','w',encoding='utf-8') as of:
    yaml.dump(data, of, default_flow_style=False, allow_unicode=True)
```

```
# CSV file toolhire.csv
ItemID,Name,Description,Owner,Borrower,DateLent,DateReturned
1,LawnMower,Small Hover mower,Fred,Joe,4/1/2012,4/26/2012
2,LawnMower,Ride-on mower,Mike,Anne,9/5/2012,1/5/2013
3,Bike,BMX bike,Joe,Rob,7/3/2013,7/22/2013
4,Drill,Heavy duty hammer,Rob,Fred,11/19/2013,11/29/2013
5,Scarifier,"Quality, stainless steel",Anne,Mike,12/5/2013,
6,Sprinkler,Cheap but effective,Fred,,,

# Reading the file using list
import csv
with open ('toolhire.csv' ) as th:
  toolreader = csv.reader(th)
  print (list(toolreader))

O/P: (note, we lost the double quotes):
[['ItemID', 'Name', 'Description', 'Owner', 'Borrower',
DateLent', 'DateReturned'],
```

```
# Writing the file using list
# writer.writerow() returns the no of characters
written , ignore that.
# note that we got the double quote back

import csv
items = [   # this is a list of lists
['2','Lawnmower','Ride-on
mower','Mike','$370','Fair','2012-04-01'],
['3','Bike','BMX bike','Joe','$200','Good','2013-03-
22'],
['4','Drill','Heavy duty
hammer','Rob','$100','Good','2013-10-28'],

with open('tooldesc.csv','w', newline='') as tooldata:
  toolwriter = csv.writer(tooldata)
  for item in items:
    toolwriter.writerow(item)
```

```
# Reading the csv file using Dict
with open('tooldesc.csv') as th:
  rdr = csv.DictReader(th)

for item in rdr: print(item)

O/P:
{'DateReturned': '4/26/2012', 'Description':
'Small Hover mower', 'Owner': 'Fred', 'ItemID':
'1', 'DateLent': '4/1/2012',
'Name': 'LawnMower', 'Borrower': 'Joe'}
{'DateReturned': '1/5/2013', 'Description':
'Ride-on mower', 'Owner': 'Mike', 'ItemID': '2',
'DateLent': '9/5/2012',
'Name': 'LawnMower', 'Borrower': 'Anne'}
```

```
# Adding Label to csv file
import csv
fields = ['ItemID', 'Name', 'Description', 'Owner', 'Price',
'Condition', 'DateRegistered']

with open('tooldesc2.csv') as td_in:
  rdr = csv.DictReader(td_in, fieldnames = fields)
  items = [item for item in rdr]

with open('tooldesc3.csv', 'w', newline='') as td_out:
  wrt = csv.DictWriter(td_out, fieldnames=fields)
  wrt.writeheader()
  wrt.writerows(items)
```

```
# Find all items rented by Fred
items_rented = []
with open('toolhire.csv') as th:
  rdr = csv.DictReader(th)

  for item in rdr:
    if item['Borrower'] == 'Fred':
       items_rented.append(item['Name'])

  #items = [item for item in rdr]

#[item['Name'] for item in items if item['Owner'] ==
'Fred']

O/P:
['LawnMower', 'Sprinkler']

# example: reading from tab delimited csv (BAD!!)
..
try:
  with open(fname) as fh:
    reader = csv.reader(fh, dialect=csv.excel_tab)
    header = reader.next()
    data = [row for row in reader]
except csv.Error as e:
  print("blah ..")
  sys.exit(-1)

if header:
  print(header)
for datarow in data:
  print(datarow)
```

```
# Reformat Date and write to csv file
import csv
from datetime import datetime

def convertDate(item):
  if item[-1] == "DateReturned":
     return item
  else:
     theDate = item[-1]
     dateObj = datetime.strptime(theDate,'%Y-%m-%d')
     dateStr = datetime.strftime(dateObj,'%m/%d/%Y')
     item[-1] = dateStr
     return item

with open('tooldesc.csv') as td:
    rdr = csv.reader(td)
    items = list(rdr)

items = [convertDate(item) for item in items]

with open('tooldesc2.csv', 'w', newline='') as td:
    wrt = csv.writer(td)
    for item in items:
      wrt.writerow(item)
```

```
# ConfigParser

[DEFAULT]
Option1=value1

[SECTION1]
Option2=value2
Option3=value3

[SECTION2]
Option4=value4

import configparser as cp
conf = cp.ConfigParser()

conf['DEFAULT'] = {'lending_period': 0, 'max_value': 0}

conf['Fred'] = {'max_value': 200}
# Fred's a bit rough with things!

conf['Anne'] = {'lending_period': 30}
# She is a bit forgetful sometimes
with open('toolhire.ini', 'w') as toolhire:
  conf.write(toolhire)

del(conf) # get rid of the old one


# My ref code on config parser

import configparser as cp

def read_config(apptype, file=CFGFILE):

conf = cp.ConfigParser()
if conf.read(file):
  defconfig = {x:y for x,y in conf.items('default')}
  appconfig = {k:v for k,v in conf.items(apptype)}
  defconfig.update(appconfig)
  netconfig = defconfig

return netconfig

# XML file: log-file-prob-6.xml

<?xml version="1.0" encoding="UTF-8"?>
<tomcat-users>
  <user name="tomcat" password="tomcat" roles="tomcat" />
  <user name="role1" password="tomcat" roles="role1" />
  <user name="both" password="tomcat" roles="tomcat,role1" />
</tomcat-users>

from xml.etree import ElementTree as et
tcurse=et.parse('logs/log-file-prob-6.xml')

for item in tcurse.findall('./user'):
  print(item.tag + '>>>' + str(item.text) + ' >>> ' +
  str(item.attrib))

#for e in tcurse.findall('./user'):
#   print(e.attrib)

# tcurse.findall('./tomcat-users') <<< no putput

O/P: user>>>None >>> {'name': 'tomcat', 'password': 'tomcat',
'roles': 'tomcat'}
..

print(e.attrib['roles']) O/P: tomcat,role1  ..

print(e.tag)      O/P: user (4 lines each line is user)

print(e.text)     O/P: None (as there is no text )
```

```
# urllib to fetch an url

#!/usr/bin/env python

import urllib.request
import urllib.error

url = 'http://www.google.com'

try:
  # Open the URL and get a response object
  with urllib.request.urlopen(url) as response;
    content_string =
      response.read().decode('utf-8')

    # Print the HTML content
    print(content_string)

except urllib.error.HTTPError as e:
  # Handle specific HTTP errors (e.g., 404 Not Found)
  print(f'HTTP error: {e.code} {e.reason}')

except urllib.error.URLError as e:
  # Handle URL errors (e.g., network issues, invalid URL)
  print(f'URL error: {e.reason}')

# urllib and HTML together

<html>
<head>
<TITLE >Profile: Dionysus</title  / >
</head>
<body bgcolor="yellow">
<center>
<br><br>
<img src="/static/dionysus.jpg" />
<h2>Name: Dionysus</h2>
<img src="/static/grapes.png"><br><br>
Hometown: Mount Olympus
<br><br>
Favorite animal: Leopard <br>
<br>
Favorite Color: Wine
</center>
</body>
</html>

#!/usr/bin/env python

from urllib.request import urlopen
from bs4 import BeautifulSoup

url = "http://olympus.realpython.org/profiles/dionysus"

page = urlopen(url)
html = page.read().decode('utf-8') # print(html)

soup = BeautifulSoup(html,'html.parser')

print(soup.title) # O/P: <title>Profile: Dionysus</title>

print(soup.title.string)  # O/P: Profile: Dionysus

print(soup.get_text())
# Profile: Dionysus
#
# Name: Dionysus
# ..

print(soup.find_all("img"))
# [<img src="/static/dionysus.jpg"/>, <img
src="/static/grapes.png"/>]

print(soup.find_all('img', src="/static/dionysus.jpg"))
# [<img src="/static/dionysus.jpg"/>]

for item in soup.find_all('img'):
  print(item.name)   # img
  print(item['src']) # img.src = /static/dionysus.jpg
                     # and /static/grapes.png
```

Reference code blocks

```
## Reading from a file
f = open("readfile.txt", "r")

#Reads one line until '\n'
print f.readline()

f.close()

# example
f= open ("readfile.txt", "r")
myList = []

for line in f:
    myList.append(line)
    print myList

f.close()

# example
a = open ("writefile.txt", "r")
print a.read()
a.close()

read([n])  # reads file as single string including '\n'
readline() # reads one line including '\n'
readlines([n]) # returns a list with each line as element
including '\n'
```

```
## Writing to a file
f = open ("writefile.txt", "w")
f.write("This is first line\n")
f.close()


# example: reading in buffer size and writing to
file
def main():
    buffersize = 50000
    infile = open('bigfile.txt', 'r') || 'rb'
    outfile = open('new.txt', 'w')    || 'wb'
    buffer = infile.read(buffersize)
    while len(buffer):
        outfile.write(buffer)
        buffer = infile.read(buffersize)

    print()
    print('Done')

    infile.close()
    outfile.close()

if __name__ == "__main__": main()

# rb || wb : read write in binary mode
```

```
# read from stdin until you type exit
import sys

for line in sys.stdin:
    if line.strip('\n') == 'exit':
        break
    print(f'Processing from stdin: ******
{line}*****')

O/P:
hello
Processing from stdin: ****** hello
*****
hello1
Processing from stdin: ****** hello1
*****
exit
```

```
# Simple Exception
var1 = '1'
try:
        var1 = var1 + 1
except:
        print var1, " is not a
number"
print var1

# example
var1 = '1'
try:
        var2 = var1 + 10
except:
        var2 = int( var1 ) + 10

print "var2 is ", var2
```

```
# Raising user defined exception
def main():
    try:
      for line in
readfile('xlines.doc'):
        print(line.strip())
    except IOError as e:
        print('cannot read file:', e)
    except ValueError as e:
        print('bad filename', e)

def readfile(filename):
    if filename.endswith('.txt'):
        fh = open(filename)
        return fh.readlines()
    else:
        raise ValueError('Filename
must end with .txt')

if __name__ == "__main__": main()
```

```
# classcalc.py
class Calculator(object):
        # define class to simulate a simple calculator
        def __init__(self):
                #start with zero
                self.current = 0
        def add(self, amount):
                #add number to current
                self.current += amount
        def getCurrent(self):
return self.current

classex1.py
############
from classcalc import *
myBuddy = Calculator()
myBuddy.add(2)
print myBuddy.getCurrent()
```

```
# find fibonaci numbers

class Fibonacci():
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def series(self):
        while(True):
            yield(self.b)
            self.a, self.b = self.b, self.a + self.b

f = Fibonacci(0, 1)
for r in f.series():
    if r > 100: break
    print r
```

```
# Generator: find primes upto 100.
def isprime(n):
    if n == 1:
        return False
    for x in range(2, n):
        if n % x == 0:
            return False
    else:
        return True

def primes( n = 1):
    while(True):
        if isprime(n): yield n
        n += 1

for n in primes():
if n > 100: break print (n)
```

```
# Threading
from threading import Thread
import time

def timer(name, delay, repeat):
  print "Timer: " + name + "
Started"

  while repeat > 0 :
    time.sleep(delay)
    print name + ": " +
      str(time.ctime(time.time()))
      repeat -= 1
    print "Timer: " + name +
     "Completed"

def main():
  t1 = Thread(target=timer,
args=("Timer1", 1, 5))
  t2 = Thread(target=timer,
args=("Timer2", 2, 5))

  t1.start()
  t2.start()
  print "Main Completed"

if __name__ == "__main__": main()
```

```
# Threading with Lock(): timer-lock.py
import threading
import time

tLock = threading.Lock()

def timer(name, delay, repeat):
  print "Timer: " + name + " Started"
  tLock.acquire()
  print name + " has acquired the lock"
  while repeat > 0 :
    time.sleep(delay)
      print name + ": " +
        str(time.ctime(time.time()))
        repeat -= 1
      print name + " is releasing the lock"
       tLock.release()
       print "Timer: " + name + "Completed"

def main():
  t1 = threading.Thread(target=timer,
args=("Timer1", 1, 5))
  t2 = threading.Thread(target=timer,
args=("Timer2", 2, 5))

  t1.start()
  t2.start()
  print "Main Completed"

if __name__ == "__main__": main()
```

```
# file.tell() Tells current position bytes
# Hello, how are you

#!/usr/bin/env python

import sys

filename = sys.argv[1]
bufsize = int(sys.argv[2])

fh_in = open(filename)
buf = fh_in.read(bufsize)

print("Read |{0}| and current position is
{1}".format(buf,fh_in.tell()))

fh_in.close()

## Example
fp.read(8)

# Print the position of handle
print(fp.tell())   O/P: 8

# Example
fp = open("sample2.txt", "wb")
print(fp.tell())       O/P: 0

# Writing to file, WATCH this!
fp.write(b'1010101')

print(fp.tell())       O/P: 7

# Closing file
fp.close()

# Example, 1=from current position,2=from end
f.seek(-10, 2)

# prints current position
print(f.tell())

# Converting binary to string and printing
print(f.readline().decode('utf-8'))
f.close()
```

```
# file.seek() moves the pointer to the position fseek-ftell.log:
Code is like humor. When you have to explain it, it's bad.
#seek(+-N,0|1|2), 0 default, 1 from current position, 2 from end
#!/usr/bin/env python

import sys
import os

filename = sys.argv[1]
bufsize = int(sys.argv[2])
seek_pos = int(sys.argv[3])

fh = open (filename,'rb')
buf = fh.read(bufsize)
current_pos = fh.tell()

print("Read |{0}| and current position is:
{1}".format(buf,current_pos))
print("Moving current position by {0} and reading again by
{1}".format(seek_pos,bufsize))

fh.seek(seek_pos,1)
buf = fh.read(bufsize)

print("Read |{0}| and current position is:
{1}".format(buf,current_pos))

fh.seek(seek_pos,1)
buf = fh.read(bufsize)

print("Read |{0}| and current position is:
{1}".format(buf,current_pos))

fh.close()

O/P: fseek-ftell.py logs/fseek-ftell.log  5 3
Read |b'Code '| and current position is: 5
Moving current position by 3 and reading again by 5
Read |b'like '| and current position is: 5
Read |b'or. W'| and current position is: 5
```

```
# reading a growing file
#!/usr/bin/env python

import time
import sys
import os

filename = sys.argv[1]

if not os.path.isfile(filename):
    print("invalid file name or file doesnot exist")

with open(filename) as fh:
    filesize = os.stat(filename)[6]
    fh.seek(filesize)   # move to end of file

    while True:
        where = fh.tell()
        line = fh.readline()
        if not line:
            time.sleep(1)
            fh.seek(where)
        else:
            print(line)
```

```
# Find file permissions
import os

p=os.stat('th2.csv')[0]
octal_perm=oct(p)
file_perm=octal_perm[-3:]

print(file_perm)
```

```
# Compare files in two dirs. dir1 and dir2
#!/usr/bin/env python

import sys
import filecmp as fc

def main():
  dir1 = sys.argv[1]
  dir2 = sys.argv[2]

  comparison = fc.dircmp(dir1, dir2)
  common_files = ', '.join(comparison.common)
  left_only = ', '.join(comparison.left_only)
  right_only = ', '.join(comparison.right_only)

  with open('compare_file_report.txt','w') as fh:
    fh.write('Common files: ' + common_files + '\n')
    fh.write('Left only files: ' + left_only + '\n')
    fh.write('Right only files: ' + right_only + '\n')


if __name__ == "__main__":
  sys.exit(main())
```

```
# Send df -h output via email
#!/usr/bin/env python

import smtplib
import subprocess

FROM = "server1@example.com"
TO = "santanu@santanuc.com"

df_pipe = subprocess.Popen('df -h', shell=true,
stdout=subprocess.PIPE)
df_result = df_pipe.communicate()[0]

msg = "Sub: df -h output " + df_result
server = smtplib.SMTP('localhost')
server.sendmail(FROM,TO,msg)
server.quit()

# Place the file in /etc/cron.daily/nightly_disk_report.py
```

```
# Regex-tricks
#!/usr/bin/env python

# \b word blundary || \w matches alpha numeric chars, no space (\W is complement)
# \d digits (\D is complement)
# if a character is used to from regex is a part of the match then it needs to be escaped
# e.g. to match [abcd] >>> \[(.*?)\]
# e.g. to match \\ten >>> r'\\ten'

# regex.sub || regex.findall || regex.finditer || regex.match || regex.search

import re

s1 = '  foo'
s2 = 'This is {{word}} inside curly brackets {{another word}} and finally {{boo!}}'
s3 = 'time tame tune tint tire'
s4 = "This is the the string I want"
s5 = "1 2 34 5"

# s1: regex.search Vs regex.match
regexp = re.compile(r'foo')
regexp.search(s1).group(0) # matches foo, watch the group#

regexp = re.compile(r'(foo)')
regexp.search(s1).group(1) # matches foo, watch the group#

regexp.match(s1)  # no match as match matches from the begining

regexp = re.compile(r'(foo)')
regexp.match(s1,pos=2).group(1) # matches as we told to starh matching at position #2

# s2: extract only words inside curly races (no greedy match example), code with findall and finditer
regexp = re.compile(r'{{(.*?)}}')
regexp.search(s2)
for item in regexp.findall(s2): print(item)

for item in regexp.finditer(s2): print(item.group(1))
# s3: extract into this list ['time', 'tame', 'tune', 'tint tire']

regexp = re.compile(r'\bt.*e\b') # does greedy matching entire sting is matched
regexp.findall(s3)

regexp = re.compile(r'\bt.*?e\b') # placing a ? doesnot do greedy matching
regexp.findall(s3)

for item in regexp.finditer(s3): print(item.group(0))

# s4: Replace two 'the' with one

regexp = re.compile(r'the the')
regexp.sub('the',s4)

# s5: return a string with "1.0 2.0 34.0 5.0"

regexp = re.compile(r'(\d+)')

items = [ item + '.0' for item in regexp.findall(s5) ]
print(','.join(items))

# log file parse tricks
doc_root_regexp.sub(r'\1 %s' %dr, line)
```

```
# Regex: Surname Name middlename: Phone – method-1

#!/usr/bin/env python

# Input file name: logs/name-and-phone-record.log
# for each line in the file display in the following way
# First name: .. Last name: .. Middle name: .. Phone no: ..
# Middle name can be blank, phone no can be missing forst 3
digits

import sys
import re

file_name = sys.argv[1]

regexp = re.compile(
     r"(?P<first>[a-zA-Z]+)"
     r",\s+(?P<last>[a-zA-Z]+)"
     r"\s+(?P<middle>([a-zA-Z]+)?)"
     r":\s+(?P<phone>(\d\d\d)?-\d\d\d\d\d\d\d)"
     )

with open(file_name,'r') as fh:
    for line in fh:
        result = regexp.search(line)

        if result:
            first = result.group('first')
            last = result.group('last')
            middle = result.group('middle')
            if middle is None:
                print("No middle name found")
            phone = result.group('phone')

            print("First name: {0} Last name: {1} Middle
name: {2} Phone no: {3}".format(first,last,middle,phone))
```

```
# Regex: Surname Name middlename: Phone – method-2

regexp = re.compile(r"([a-zA-Z]+)"
                    r",\s+([a-zA-Z]+)"
                    r"\s+(([a-zA-Z]+)?)"
                    r":\s+((\d\d\d)?-\d\d\d\d\d\d\d)"
                    )

with open(file_name,'r') as fh:
    for line in fh:
        result = regexp.search(line)

        if result:
            first = result.group(1)
            last = result.group(2)
            middle = result.group(3)
            if not middle:
                print("No middle name found")
                middle = "XXXX"
            phone = result.group(4)

            print("First name: {0} Last name: {1}
Middle name: {2} Phone no:
{3}".format(first,last,middle,phone))
```

```
# Regex: Find files matching a pattern

#!/usr/bin/env python

# This program finds files with matching patterns

import sys
import os
import re


def find_files(pattern, base='.'):
    '''
    display help test here
    '''
    regexp = re.compile(r"" + pattern)
    matched_files = []

    for root, dirs, files in os.walk(base):
        for file in files:

            if regexp.match(file):
                matched_files.append(os.path.join(root,file))

    return matched_files

def main():
    pattern = sys.argv[1]
    base = sys.argv[2]
    #print(find_files(pattern, base))
    #['./polymorphism-1.py', './polygon-triangle-inheritance.py', './polymorphism-2.py']
    print(find_files('pol', '.'))
    #['./polymorphism-1.py', './polygon-triangle-inheritance.py', './power-of-two-iterator.py', './power-of-two-
generator.py', './polymorphism-2.py']
    # the following matches 0 or more l characters, thats the gotchaa!
    print(find_files('pol*', '.'))

if __name__ == "__main__":
    sys.exit(main())
```

```
# Regex: Return host, status, bytes sent and write to a csv file
127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0" 200 2326 "http://www.example.com/start.html"
"Mozilla/4.08 [en] (Win98; I ;Nav)"
127.0.0.5 - jane [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0" 200  1326 "http://www.example.com/start.html"
"Mozilla/4.08 [en] (Win98; I ;Nav)"
127.0.0.9 - bob [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0" 400  0 "http://www.example.com/start.html" "Mozilla/4.08
[en] (Win98; I ;Nav)"

#!/usr/bin/env python

import sys
import os
import re
import csv
import argparse

prog_name = sys.argv[0]

def example_usage(prog):
    '''
    help
    '''
    helpstring = """
    Example:
    ========
    1. To display help:
       prog_name -h
    2. To generate report:
       prog_name -i <input_file_name> [-o <output_file_name>]
    """

    return helpstring.replace('prog_name', prog)

def read_file(filename):
    with open(filename,'r') as fh:
        for line in fh:
            yield line

def main():
    parser = argparse.ArgumentParser(description = "Generates csv report file from input logfile",
                                     epilog=example_usage(prog_name),
                                     formatter_class=argparse.RawDescriptionHelpFormatter)

    parser.add_argument('-i', '--input', action='store', help="Provide input file name")
    parser.add_argument('-o', '--output', action='store', help="Provide output file name")

    args = parser.parse_args()
    if args.input:
        infile = args.input
        if args.output:
            outfile = args.output
        else:
            d,f = os.path.split(infile)
            f1,e1 = os.path.splitext(f)
            outfile = os.path.join(d,f1+'.csv')
    else:
        print("you must provide input file name")
        example_usage()
        sys.exit(1)

    regexp = re.compile(r'(\d+.\d+.\d+.\d+)\s+-\s+.*?\s+\[.*?\]\s+".*?"\s+(\d+)\s+(\d+).*')
    fileds = ["Remote Host", "Status", "Bytes sent"]
    with open (outfile, 'w') as fh:
        writer = csv.DictWriter(fh, fieldnames=fileds)
        writer.writeheader()

        for line in read_file(infile):
            result = regexp.search(line)
            if result:
                remote_host = result.group(1)
                status = result.group(2)
                bytes_sent = result.group(3)
                line_dict = {"Remote Host": remote_host, "Status": status, "Bytes sent": bytes_sent}
                writer.writerow(line_dict)

if __name__=="__main__":
    sys.exit(main())
```

```
# Sandia no-regex, argparse, generator

#!/usr/bin/env python

import sys
import os
import argparse

progname = sys.argv[0]

def example_usage(prog):
    '''
    help
    '''
    helpstring = """
    Example usage:
    ==============
    1. Display help:
       prog_name -h
    2. Analyze logs:
       prog_name -a -i <log_file_name> [-o <repoprt_file> ]
    """
    return helpstring.replace("prog_name", prog)

def read_file(file_name):
    with open (file_name,'r') as fh:
        for line in fh:
            yield line


def write_report(output_file,host_cnt):
    with open(output_file,'w') as fh:
        for k,v in host_cnt.items():
            fh.write(k + ' ' + str(v) + '\n')

def main():
    host_cnt = {}

    parser = argparse.ArgumentParser(description="Analyze the log file",
                                     epilog=example_usage(progname),
                                     formatter_class=argparse.RawDescriptionHelpFormatter)

    parser.add_argument('-a', '--analyze', action='store_true', help="Analyze logs")
    parser.add_argument('-i', '--infile', action='store', help="Provides the input file name")
    parser.add_argument('-o', '--outfile', action='store', help="Provides the output report file name")

    args = parser.parse_args()

    if args.infile:
        if args.analyze:
            input_file = args.infile
            if args.outfile:
                output_file = args.outfile
            else:
                d,f = os.path.split(input_file)
                output_file = os.path.join(d,'records_'+ f)
    else:
        print("You  must provide the input file")
        sys.exit(1)

    for line in read_file(input_file):
        host_name = line.strip('\n').split()[0]
        if host_cnt.get(host_name):
            host_cnt[host_name] = host_cnt[host_name] + 1
        else:
            host_cnt[host_name] = 1

    write_report(output_file,host_cnt)

if __name__ == "__main__":
    sys.exit(main())
```

```
# Regex: apache config file parse
# pdp-4.py logs/log-file-prob-4.conf local2:80 /tmp

#!/usr/bin/env python

import sys
import re
import argparse

def read_config(filename,vhost,doc_root):
    regexp_vhost_start = re.compile(r'<VirtualHost\s+(.*?:80)')
    regexp_currdocroot = re.compile(r'(DocumentRoot)\s+(.*)')
    regexp_vhost_end = re.compile(r'</VirtualHost>')

    vhost_start = 'no'
    vhost_end = 'no'

    with open(filename,'r') as fh:
        for line in fh:
            result = regexp_vhost_start.search(line)
            if result:
                curr_doc_root = result.group(1)
                # if it matches local2:80
                if curr_doc_root == vhost:
                    vhost_start = 'yes'

            result = regexp_currdocroot.search(line)
            if result and vhost_start == 'yes':
                d = result.group(1)
                line = d + " " + doc_root
                #line = regexp_currdocroot.sub("\1 %s" %doc_root)

            if regexp_vhost_end.search(line) and vhost_start == 'yes':
                vhost_end = 'yes'

            yield line.strip('\n')

def main():
    conf_file = sys.argv[1]
    vhost = sys.argv[2]
    doc_root = sys.argv[3]
    for line in read_config(conf_file,vhost,doc_root):
        print(line)

if __name__ == '__main__':
    sys.exit(main())
```

```
# ping IPs in /etc/hosts with thread,

#!/usr/bin/env python

import sys
import re
import subprocess
from threading import Thread
import queue
from IPy import IP

result = {}

def pinger(i,q):
  while True:
      ip = q.get()
      ret = subprocess.call("ping -c 1 %s" %ip,
       shell=True, stdout=open('/dev/null', 'w'),
       stderr=subprocess.STDOUT)
      if ret == 0:
          #print("IP %s is alive" %ip)
          result.update({ip:"is live"})
      else:
          result.update({ip:"is NOT alive"})

      q.task_done()


def main():

  q = queue.Queue()

  regexp = re.compile(r'::|127|255|#|(\s+)')
  myips = []

  num_threads = sys.argv[1]

  with open('/etc/hosts','r') as fh:
    for line in fh:
      if not regexp.match(line):
          myips.append(line.strip('\n').split(' ' )[0])

  for ip in myips:
    q.put(ip)


  for i in range(int(num_threads)):
    th = Thread(target=pinger, args=(i,q))
    th.daemon = True
    th.start()

  q.join()

  for k,v in result.items():
    print(k,v)

if __name__ == "__main__":
    sys.exit(main())
```

```
# ping IPs in /etc/hosts with multiprocessing

#!/usr/bin/env python

'''
This will not add the results to common dict
'''

import sys
import subprocess
import re
import multiprocessing as mp
import queue
from IPy import IP

results = {}

def pinger(i,q):
    while True:
        if q.empty():
            sys.exit()
        ip = q.get()

        ret = subprocess.call("ping -c 1 %s" %ip,
shell=True, stdout=open('/dev/null','w'),
stderr=subprocess.STDOUT)
        if ret == 0:
            results.update({ip:'is Alive'})
        else:
            results.update({ip:'is NOT Alive'})

def main():
    num_threads = sys.argv[1]
    q = mp.Queue()
    myips = []

    regexp = re.compile(r'#|::|127|255|(\s+)')

    with open('/etc/hosts','r') as fh:
        for line in fh:
            if not regexp.match(line):
                myips.append(line.strip('\n').split('
')[0])

    for ip in myips:
        q.put(ip)

    for i in range(int(num_threads)):
        worker = mp.Process(target=pinger, args=[i,q])
        worker.start()

    # This will not work as process do not share data
    worker.join()
    for k,v in results.items():
        print(k,"==>",v)


if __name__ == "__main__":
    sys.exit(main())
```

```
# ping-sweep.py using multiprocessing  & update a dict

#!/usr/bin/env python

'''
This will accumulate results from all processes into one
Dict
'''

import sys
import subprocess
import re
import multiprocessing as mp
import queue.  # not used, we use Queue from
multiprocessing lib
from IPy import IP

def pinger(ip,L):
  ret = subprocess.call("ping -c 1 %s" %ip, shell=True,
stdout=open('/dev/null','w'), stderr=subprocess.STDOUT)
  if ret == 0:
     L.update({ip:'is Alive'})
  else:
     L.update({ip:'is NOT Alive'})

def main():
  num_threads = sys.argv[1]
  q = mp.Queue()
  myips = []

  manager = mp.Manager()
  results = manager.dict()


  regexp = re.compile(r'#|::|127|255|(\s+)')

  with open('/etc/hosts','r') as fh:
    for line in fh:
      if not regexp.match(line):
        myips.append(line.strip('\n').split(' ')[0])


    pool = mp.Pool(processes=int(sys.argv[1]))

    for ip in myips:
       pool.apply_async(pinger, args=(ip, results))


    pool.close()
    pool.join()

    for k,v in results.items():
      print(k,"==>",v)


if __name__ == "__main__":
  sys.exit(main())
```

```
# Pandas find p99 percentile latency: logs/latency.csv
request_id,start_time,end_time
1,2025-01-01 10:00:00.123,2025-01-01 10:00:00.456
2,2025-01-01 10:00:01.400,2025-01-01 10:00:02.100
3,2025-01-01 10:00:02.500,2025-01-01 10:00:02.600
4,2025-01-01 10:00:03.000,2025-01-01 10:00:03.800

#!/usr/bin/env python

import pandas as pd

csv_filename = 'logs/latency.csv'

def analyze_latency(csv_filename):
  df = pd.read_csv(csv_filename,
parse_dates=['start_time', 'end_time'])

  df['latency'] = df['end_time'] - df['start_time']
  df['latency_seconds'] = df['latency'].dt.total_seconds()

  print("--- Individual Latency Numbers (in seconds) ---")
  print(df['latency_seconds'])

  print("\n--- Latency Statistics (in seconds) ---")
  print(df['latency_seconds'].describe())

  print(f"\nAverage Latency:
{df['latency_seconds'].mean():.4f} seconds")
  print(f"Median Latency:
{df['latency_seconds'].median():.4f} seconds")
  print(f"Maximum Latency:
{df['latency_seconds'].max():.4f} seconds")

  p99_latency = df['latency_seconds'].quantile(0.99)
  print(f"99th Percentile Latency: {p99_latency:.4f}
seconds")

analyze_latency(csv_filename)

O/P:
--- Individual Latency Numbers (in seconds) ---
0    0.333
1    0.700
2    0.100
3    0.800
Name: latency_seconds, dtype: float64

--- Latency Statistics (in seconds) ---
count    4.000000
mean     0.483250
std      0.324944
min      0.100000
25%      0.274750
50%      0.516500
75%      0.725000
max      0.800000
Name: latency_seconds, dtype: float64

Average Latency: 0.4833 seconds
Median Latency: 0.5165 seconds
Maximum Latency: 0.8000 seconds

99th Percentile Latency: 0.7970 seconds
```

```
# More examples
d={
 "Name": ["John", "Sarah", "Emily", "Michael", "Alex"],
 "Age": [25, 30, None, 42, 28],
 "Salary": [50000, None, 35000, None, 60000 ]
}

df2=pd.DataFrame(d)
df2           # Prints NaN for None
#      Name   Age   Salary
#0     John  25.0  50000.0
#1    Sarah  30.0      NaN
#2    Emily   NaN  35000.0
#3  Michael  42.0      NaN
#4     Alex  28.0  60000.0

df2.index     # RangeIndex(start=0, stop=5, step=1)
df2.shape     # (5, 3)
df2.columns   # Index(['Name', 'Age', 'Salary'],
dtype='object')

# slicing rows by label || by index
df2.loc[1:3, ['Name','Age']] # df2.iloc[1:3, [0,1]]
#      Name   Age
#1    Sarah  30.0
#2    Emily   NaN
#3  Michael  42.0

# Get a cell value,
df2.at[1, 'Name']  #  O/P: Sarah

# Drop a column, This prints only Name and Salary
df2.drop('Age', axis=1)

# fill missing value with 0)
df2.fillna(0)

# concat, merge, join
#logs/latency.csv
#request_id,start_time,end_time
#1,2025-01-01 10:00:00.123,2025-01-01 10:00:00.456
#2,2025-01-01 10:00:01.400,2025-01-01 10:00:02.100
#3,2025-01-01 10:00:02.500,2025-01-01 10:00:02.600
#4,2025-01-01 10:00:03.000,2025-01-01 10:00:03.800

#logs/latency-b.csv
#request_id,start_time,end_time
#1,2025-01-01 10:00:00.123,2025-01-01 10:00:00.456
#2,2025-01-01 10:00:01.400,2025-01-01 10:00:02.100
#3,2025-01-01 10:00:02.500,2025-01-01 10:00:02.600
#4,2025-01-01 10:00:03.000,2025-01-01 10:00:03.800

df10 = pd.read_csv('logs/latency.csv')
df20 = pd.read_csv('logs/latency-b.csv')

df201 = pd.concat([df10,df20], ignore_index=True)

df100 = pd.merge(df10,df20,how='left', on='request_id')
#   request_id            start_time_x
end_time_x             start_time_y              end_time_y
#0           1  2025-01-01 10:00:00.123  2025-01-01
10:00:00.456  2025-01-01 10:00:00.125  2025-01-01 10:00:00.556
#1           2  2025-01-01 10:00:01.400  2025-01-01
10:00:02.100  2025-01-01 10:00:01.409  2025-01-01 10:00:02.200
..

df101 = pd.concat([df10,df20], axis=1)
#   request_id               start_time
end_time  request_id               start_time
end_time
#0           1  2025-01-01 10:00:00.123  2025-01-01
10:00:00.456           1  2025-01-01 10:00:00.125  2025-01-01
10:00:00.556
#1           2  2025-01-01 10:00:01.400  2025-01-01
10:00:02.100           2  2025-01-01 10:00:01.409  2025-01-01
10:00:02.200
..

# not working
#df102 = df10.join(df20.set_index('request_id'),
on='request_id')
```

```
# Pandas More examples

#!/usr/bin/env python

import pandas as pd

#{
# "Name": ["John", "Sarah", "Emily", "Michael", "Alex"],
# "Age": [25, 30, 18, 42, 28],
# "Gender": ["M","F","F","M","M"],
# "Salary": [50000, 80000, 35000, 120000, 60000 ],
# "Country": ["USA", "Canada", "USA", "USA", "Canada" ]
#}

# read and write from json,csv.excel | read_csv,read_excel |
to_csv,to_excel
df1 = pd.read_json('logs/p104.json')
df1.to_json('logs/p104-out.json')

# inplace changes the df1
df1.sort_values(by=['Age','Salary'],inplace=True,ignore_index=True)
df3 = df1.query('Age >29 and Salary >50000')

# index is random
df1.sort_index(inplace=True)

df1.groupby(['Country'])[['Salary','Age']].mean()
#               Salary        Age
#Country
#Canada   70000.000000  29.000000
#USA      68333.333333  28.333333

# apply function to column
df1.groupby(['Country'])[['Salary','Age']].mean().apply(lambda x:
x.max() - x.min())
#Salary    1666.666667
#Age          0.666667
#dtype: float64

df1.groupby(['Country'])[['Salary','Age']].quantile(.99)
#           Salary    Age
#Country
#Canada    79800.0  29.98
#USA      118600.0  41.66
```

```
# Pizzacol generator expression
with open('..' 'r') as fh:
  pizza_col = (line.split()[5] for line in fh)
  pizza_per_hr = (int(x) for x in pizza_col if x!='N/A')
  print("total pizza", sum(pizza_per_hr))

# Signal / trap
#!/usr/bin/env python

import signal
import time

def signal_handler(signum, frame):
    if signum == 2:
        print("You pressed CRTL+ C")
    else:
        print("it was not CRTL + C")


signal.signal(signal.SIGINT, signal_handler)
signal.signal(signal.SIGQUIT, signal_handler)

n = 1
while True:
    print(".")
    time.sleep(1)
    n += 1
    if n == 10:
        break
```

```
# Password generator
#!/usr/bin/env python

import random

# add all lowercase and uppercase alphabets
letters = ['a','b','c','d','e']
numbers = ['0','1','2','3','4']
special_chars = ['@','!','^','-','%']

l = int(input("Provide number of letters: "))
n = int(input("Provide number of numers: "))
c = int(input("Provide number of special chars: "))

pwd_list = []

for x in range(l):
    pwd_list.append(random.choice(letters))

for x in range(n):
    pwd_list.append(random.choice(numbers))

for x in range(c):
    pwd_list.append(random.choice(special_chars))

random.shuffle(pwd_list)

password = ''.join(pwd_list)

print("Password is: ", password)
```

```
# Polymorphism Ex-1
#!/usr/bin/env python

class Parrot():
    def __init__(self):
        pass

    def fly(self):
        print("Parrot can fly")

    def swim(self):
        print("Parrot cannot swim")


class Penguine():
    def __init__(self):
        pass

    def fly(self):
        print("Penguine cann't fly")

    def swim(self):
        print("Penguine can swim")


def flying_test(animal):
    animal.fly()

peggy = Penguine()
blue = Parrot()

animals = [ peggy, blue]

for animal in animals:
    flying_test(animal)
```

```
# Polymorphism Ex-2
#!/usr/bin/env python

class Car():

    def __init__(self, name):
        self.name = name

    def drive(self):
        raise NotImplementedError("Subclass ...")

    def breaking(self):
        raise NotImplementedError("Subclass ...blah blah")

class Sportscar(Car):

    def drive(self):
        return " drives fast"
        # print(" drives fast") dos not work as it is returning
None

    def breaking(self):
        return " breaking fast"

class Truck(Car):

    def drive(self):
        return " drives slow"

    def breaking(self):
        return " breaking"


banana_truck = Truck('Banana Truck')
orange_truck = Truck('Orange Truck')
acura_integra = Sportscar('Z3')

cars = [ banana_truck, orange_truck, acura_integra ]

for motor in cars:
    print(motor.name, motor.drive(), motor.breaking())
```

```
# Inheritence
#!/usr/bin/env python

class Polygon():

  def __init__(self,n):
     self.num_sides = n
     self.sides = [0 for i in range(self.num_sides)]

  def input_sides(self):
     self.sides = [float(input("Provide side value for side "+
str(side + 1) + ": ")) for side in range(self.num_sides)]

  def display_sides(self):
     for side in self.sides:
         print("side has value %s" %side)


class Triangle(Polygon):

  def __init__(self):
     super().__init__(3)

  def peremeter(self):
     return(sum(self.sides))


t = Triangle()
t.input_sides()
t.display_sides()

print(t.peremeter())
print(isinstance(t,Triangle))
print(issubclass(Triangle,Polygon))
```

```
# Encapsulation
#!/usr/bin/env python

class Computer():

    def __init__(self):
        self.__max_price = 900

    def get_price(self):
        return self.__max_price

    def set_price(self, price):
        self.__max_price = price


c = Computer()
# 900
print(c.get_price())

# 900
c.__max_price = 1000
print(c.get_price())

c.set_price(1000)
print(c.get_price())
```

```python
# Iterators – power of two
#!/usr/bin/env python

class PowTwo():
    def __init__(self,max=0):
        self.max = max

    def __iter__(self):
        self.n = 0
        return self

    def __next__(self):
        if self.n < self.max:
            result = 2 ** self.n
            self.n += 1
            return result
        else:
            raise StopIteration

# create an object
numbers = PowTwo(3)

# create an iterable from the object
i = iter(numbers)

# Using next to get to the next iterator element
print(next(i)) # prints 1
print(next(i)) # prints 2
print(next(i)) # prints 4
print(next(i)) # prints 8
print(next(i)) # raises StopIteration exception
```

```python
# Infinite Iterators – odd number
#!/usr/bin/env python

class OddNumber():
    def __init__(self):
        self.n = 1

    def __iter__(self):
        return self

    def __next__(self):
        result = self.n
        self.n += 2

        return result


a = OddNumber()
i = iter(a)

for x in range(10):
    print(next(i))

# Power of two using yield

def PowTwoGen(max=0):
    n = 0
    while n < max:
        yield 2 ** n
        n += 1
```

```python
# Operator overloading, Example-1
#!/usr/bin/env python

class Point():

    def __init__(self,x,y):
        self.x = x
        self.y = y

    def __str__(self):
        return "({0},{1})".format(self.x,self.y)

    def __add__(self, other):
        x = self.x + other.x
        y = self.y + other.y

        return Point(x,y)

    def __lt__(self,other):
        self_mag = self.x ** 2 + self.y ** 2
        other_mag = other.x ** 2 + other.y ** 2

        return self_mag < other_mag



p1 = Point(2,3)
print(p1)

p2 = Point(5,6)
print(p1+p2) ) # this calls the __add__() method

print((p3.x, p3.y))     # O/P: (3, 5)
Point(1,1) < Point(1,1) # O/P: False
```

```python
# Operator overloading, Example-2

class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    # overload < operator
    def __lt__(self, other):
        return self.age < other.age

p1 = Person("Alice", 20)
p2 = Person("Bob", 30)

print(p1 < p2)  # prints True
print(p2 < p1)  # prints False
```

```
# Assign attributes to object in the fly, Ex-1
#!/usr/bin/env python

class ComplexNumber():

  def __init__(self,a,b):
    self.real = a
    self.img = b

  def __str__(self):
    return "({0}+{1}i)".format(self.real,self.img)

c1 = ComplexNumber(2,3)

print(c1)

# 10
c1.attr = 10
print(c1.attr)
```

```
# overload __getattr__ and __getatrtributes__, Ex-2

#!/usr/bin/env python

class Dummy():
    pass

    def __getattr__(self,attr):
        return attr.upper()

    def __getattribute__(self, attr):
        return "I intercept everything, my way or no way"

d = Dummy()

d.does_exist = "Hello"
# Hello --> I intercept everything, my way or no way (due to
__getattribute__)
print(d.does_exist)

# DOES_NOT_EXIST --> I intercept everything, my way or no way
(due to  __getattribute__)
print(d.does_not_exist)
```

```
# Closures - example-1
#!/usr/bin/env python

def make_multiplier(n):

    def multiplier(x):
        return n * x

    return multiplier

# 12
times3 = make_multiplier(3)
print(times3(4))

# 15
times5 = make_multiplier(5)
print(times5(3))

# 30
print(times3(times5(2)))
```

```
# Closures - example-2
#!/usr/bin/env python

def print_msg(msg):

    def printer():
        print(msg)

    return printer()

a = print_msg('Hello')

# This doesnot work !!
a()
```

```
# Decorator - example-3
#!/usr/bin/env python

def make_pretty(func):
    def inner():
        print("I got decorated")
        func()

    return inner

@make_pretty
def ordinary():
    print("I am ordinary")


a = ordinary()
```

```
# Decorator - example-4
#!/usr/bin/env python

def smart_divide(func):

    def inner(a,b):
        if b == 0:
            print("Whoops! b cannot be zero")
            return

        return func(a,b)

    return inner

@smart_divide
def divide(a,b):
    return a/b

#
print(divide(4,2))
print(divide(2,22))
print(divide(4,0))
```

```
# Download file using urllib
import urllib.request

url = "https://example.com/file.txt"
output_file = "file.txt"

try:
    with urllib.request.urlopen(url) as response,
open(output_file, "wb") as out_file:
        out_file.write(response.read())
    print("Download complete!")
except Exception as e:
    print(f"Download failed: {e}")
```

```
#
```

++++++

Iterating over groups of things:
Data:
Toy Solders (1991) | R. Lee Ermey
Toy Solders (1991) | Wil Wheaton
Toy Solders (1991) | Sean Astin
Toy Story (1991) | Don Rickles
Toy Story (1991) | Tom Hanks
Toy Story (1991) | Tim Allen
Toy Story (1991) | Wil Wheaton

Expected O/P:

Toysoldier = [‘R. Lee Ermey’, ‘Wil Wheaton’, ..]
Toystory = [‘Don Rickles’, ‘Tom Hanks’, ..]

Numpy:

```
# import numpy as np
np.zeros(5)
np.ones((2,3))
np.arange(5)
np.linspace(0,1,5)
np.ful((2,2), 7)
np.eye(3)
np.random.random((2,3))
np.empty((2,2))
np.array([3,2]) | np.array([3,2],[4,5])
arr1 = np.array([1,2,3,4], dtype=int)
arr2 = arr1.astype(float)
arr1_1 = arr1[0:3]
arr1.reshape(2,2)
arr1.ravel() | arr1.flatten()
arr.swapaxes(0,1) # transpose

# arithmetic ops
arr1 + arr2 | arr1 * 3 | arr1 + 4 | arr1 * arr2

# Linear algebra
C = np.dot(A,B)
E = np.eig(A)
G,H,I = np.linalg.sdv(A)
J = np.trace(A)
K = np.linalg.det(A)
Y = np.linalg.solve(A,x)
X = np.random.randint(0,10,5)
```

```
# 2D arr
np.arr([[1,3], [4,5]])

# 3D arr is object with nested lists, x is outermost, y
is nested in x, and z is inside y. It is  3 sets of
((2x2) array). shape = 2,2,3, dim = 3
np.arr([ [[1,3],[4,5]], [[4,5],[7,9]], [[2,3],[6,7]] ])

# 4D, 3 sets of (2 sets of (2x2) array )): 3x(2x(2x2))
shape = 3,2,2,2; dim = 4
np.array(
[
  [
    [[1,5], [3,7]],
    [[5,6], [8,9]]
  ],
  [
    [[8,9], [3,2]],
    [[5,1], [5,6]]
  ],
  [
    [[1,4], [3,7]],
    [[5,6], [8,9]]
  ]
]
)

# Statistical functions <<<
np.mean(), np.median(), np.std()
```

```
# np examples
import numpy as np

np.random.seed(42)

# generate random age for 500 between 18 and 66
age = np.random.randint(18, 66, size=500)

height_inches = np.random.randint(54, 84, size=500)
height_feet = height_inches/12.0
height_meters = height_inches * 0.0254

# weight in lbs
weight = np.random.randint(100, 301, size=500)
weight_kg = weight * 0.453592
bmi = weight_kg / (height_meters ** 2)

print("Age:\n mean={:.2f}, \n median={:.2f}, \n
std={:.2f}".format(np.mean(age), np.median(age),
np.std(age)))
```

```
## axis 0 is row, axis 1 is column    <<< watch this !!

# calculate the latency/ percentile value from a csv <<<
import numpy as np

# Load the employee data CSV file into a Pandas DataFrame
df = pd.read_csv('data.csv')

# Extract the salary column for analysis
salary_data = df['salary']

# Define the desired percentiles
percentiles = [25, 50, 75]

# Calculate percentiles using numpy.percentile
percentile_values = np.percentile(salary_data,
percentiles)

print(f"Salary Percentiles {percentiles}:
{percentile_values}")
```

Pandas:

```
# import pandas as pd
# data could be a csv or Jason format, e.g.
data = { 'Name': ['John', 'Sarah', 'Emily','Michael','Alex'],
        'Age': [25, 30, 18, 42, 28],
        'Salary': [50000, 80000, 35000, 120000, 60000]}

df = pd.DataFrame(data) | pd.read_csv(<csv_file_name>)
df.head([n])| df.tail([n]) | df.info()
df.shape | df.columns | df.index
df.loc[0:3, ['Name','HireDate']] |df.iloc[1:3, [0,1]]
df.at[1,'Name'] | df.iat[1,1]
df[df.ItemID > 10]
df.query('ItemID > 9 and Name == "Santanu"')
df[‘Identifier’].is_unique # Identifier is col name

df.drop('HireDate', axis=1)
# to_drop is a list with column names

df.drop(columns=to_drop, inplace=True, axis=1)
```

```
london = pub.str.contains(‘London’)

df['Place of Publication'] = np.where(london,
'London',
   np.where(oxford,
'Oxford',
     pub.str.replace('-', ' ')))

towns_df = pd.DataFrame(<func>, columns=['State',
'RegionName'])

towns_df =  towns_df.applymap( <func> )

pd.to_datetime(date, format='%Y-%m-%d')   <<<
pd.to_datetime(time, format="%H:%M")

df['is_date_valid'] = df['date'].apply(<func>)

for col in df.columns:          <<<
```

```
df.sort_values(by='Age', inplace=True)
df.sort_values(by='Salary', ascending=False, inplace=True)
df.sort_values(by='Age', ignore_index=True, inplace=True)

df.groupby(['Country'])[['Salary','Age']].mean()

df.isnull() | df.duplicated()

df.drop_duplicates() | df.dtypes

df.fillna(df.mean()) | df.fillna(0)
df.replace({'Anshu': 'A Kundu', 'Santanu': 'S Chakrabarty'})
df.rename(columns={'Name': 'Full Name'})
df['ItemID'] = df['ItemID'].astype(float)
df['ItemID'].value_counts()

df.get_dtype_counts()
# using regex
extr = df['Date of Pub'].str.extract(r'^(\d{4|)', expand=False)
df['Date of Pub'] = pd.to_numeric(extr)
df['Date of Pub'].isnull().sum() / len(df)


# to skip first row
df1 = pd.read_csv(<csv_file_name>, header=1)
# new_name is a dict with k = old name , v = name u want
df2 = df.rename(columns = new_name, inplace = True)
```

```
    df[col].apply(<func>)

## merging two data frames df1, df2
merged_df = pd.merge(df1, df2, on='Employee')

pd.concat([df1, df2], axis=1)

df1.join(df2.set_index('Employee'), on='Employee')

df2.groupby('Employee')['Salary'].sum()

pd.pivot_table(df1, values='Employee',
index='Department', columns='Department',
aggfunc='sum')

# Statistical functions
s = pd.Series([1,2,3,4], index=['a','b','c','d'])
s.mean(), s>3
q1 = df[col_name].quantile(.50) # p50/median

daily_sales = pd.Series(np.random.randint(100,
1000, len(dates)), index= dates)

monthly_sales_mean =
daily_sales.resample('M').mean()


rolling_sales_mean =
second_half_sales.rolling(window=30).mean()
```

```
# pd examples
data = {'Name': ['John', 'Jane', 'Adam', 'Ava'],
 'Age': [25, 30, 28, 24],
 'Salary': [50000.00, 60000.00, 55000.00, 45000.00]}

df = pd.DataFrame(data)
print(df)

print (df.dtypes)

def check_datatype(value):
    if isinstance(value, str):
        return 'string'
    if isinstance(value, int):
        return 'integer'
    if isinstance(value, float):
        return 'float'
    else:
        return 'other'

for col in df.columns:
    print("\n=====\n")
    print(col, df[col].apply(check_datatype)) <<<

print(type(value))
print(isinstance(value, str))
print(isinstance(value,object)
```

```
# pd examples, validate using custom function
data = {'date': ['2022-01-01', '2022-01-02', '2022-01-035',
'2022-01-04', '2022-01-05'],
        'time': ['12:00', '13:30', '14:15', '15:00',
'1a6:30']}

df = pd.DataFrame(data)
print(df)

def validate_date_format(date):
    try:
        pd.to_datetime(date, format='%Y-%m-%d')
        return True
    except ValueError:
        return False

def validate_time_format(time):
    try:
        pd.to_datetime(time, format="%H:%M")
        return True
    except ValueError:
        return False

df['is_date_valid'] = df['date'].apply(validate_date_format)
df['is_time_valid'] = df['time'].apply(validate_time_format)

print(df)
```

```
## pd examples

#!/usr/bin/env python

import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(10,2),
columns=['Column_A', 'Column_B'])

df.plot()
df.corr()
df.cov()


# pd examples
#!/usr/bin/env python

import pandas as pd
```

```
## Calculate percentiles using pandas. <<<

# read the csv file
q1 = df[col_name].quantile(.25)
q2 = df[col_name].quantile(.50)
q3 = df[col_name].quantile(.75)
q4 = df[col_name].quantile(.99)


## Add a computed column.  <<<
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})

df['C'] = df.apply(lambda row: row['A'] * row['B'], axis=1)

OR

df.eval('C = A**2 + B**2', inplace=True)

## find time difference  <<<
```

```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)

dates = pd.date_range(start='2022-01-01', end='2022-
12-31', freq='D')

daily_sales = pd.Series(np.random.randint(100, 1000,
len(dates)), index= dates)

monthly_sales_mean = daily_sales.resample('M').mean()

plt.plot(monthly_sales_mean)
plt.title('Monthly Sales Mean')
plt.xlabel('Month')
plt.ylabel('Sales Mean')
plt.show()

first_half_sales = daily_sales.loc['2022-01-
01':'2022-06-30']

weekly_sales_sum =
first_half_sales.resample('W').sum()

plt.plot(weekly_sales_sum)
plt.title('Weekly Sales Sum(Jan-June)')
plt.xlabel('Week')
plt.ylabel('Sales Sum')
plt.show()

second_half_sales = daily_sales.loc['2022-07-
01':'2022-12-31']

rolling_sales_mean =
second_half_sales.rolling(window=30).mean()   <<<

plt.plot(rolling_sales_mean)
plt.title('Rolling Sales Mean(Jul-Dec)')
plt.xlabel('Day')
plt.ylabel('Sales Mean')
plt.show()

sales_df = pd.concat([monthly_sales_mean,
weekly_sales_sum], axis=1)
sales_df.columns = ['Monthly Sales Mean', 'Weekly
Sales Sum']
print(sales_df)
```

```
ts1 = pd.Timestamp('11:34:56.234566')
ts2 = pd.Timestamp('11:34:56.240000')
t_delta = (ts2-ts1)


## Calculate time difference  <<<
df = pd.DataFrame({
    'timestamp': ['2022-06-17 10:00:00', '2022-06-17
11:00:00', '2022-06-17 12:00:00', '2022-06-17 13:00:00'],
    'value': [1, 2, 3, 4]
})

# convert the timestamp column to datetime format
df['timestamp'] = pd.to_datetime(df['timestamp'])

# replace missing values with a default value
df['time_diff'] =
df['time_diff'].fillna(pd.Timedelta(seconds=0))


# calculate the time difference between consecutive rows
df['time_diff'] = df['timestamp'].diff()

print(df)


## More stats mean(), 99pct. <<<

df['time1'] = pd.to_datetime(df['time1'])
df['time2'] = pd.to_datetime(df['time2'])
df['time_delta'] = df['time2'] - df['time1']

print("mean: %s 99pct %s" %(df['time_delta'].mean(),
df['time_delta'].quantile(.99)))
```