In [ ]:
```
Recap
--------
int       str        list   set
float     bytes      tuple
complex   bool       dict
operators
conditional statements  - only one time
if only
if else
if elif elif
Looping statements - code block will execute more than one
  ->conditional style - while
  ->collection style  - for
```

In [ ]:
```
# 1D
list  => L=['D1','D2','D3']

tuple => T=('D1','D2','D3')

dict  => d={'K1':'D1','K2':'D2','K3':'D3'}

# MD
# list of another list/tuple/dict
L = [ ['D1','D2','D3'],('T1','T2','T3'),{'Kx':'Vx','Ky':'Vy'}]


# tuple of another list/tuple/dict
T = (['D1','D2','D3'],('T1','T2'),{'Kx':'Vx','Ky':'Vy'})

# dict of another list/tuple/dict
d={"K1":['V1','V2','V3'],'K2':('T1','T2'),'K3':{'K1':'V1','K2':'V2'}}
```

In [1]:
```
servers = [['RHL5','RHL6','RHL7','RHL8'],('Win10','Win11')]
print(type(servers),len(servers))
```

```
<class 'list'> 2
```

In [3]:
```
print(type(servers[0]),type(servers[1]))
```

```
<class 'list'> <class 'tuple'>
```

In [4]:
```
servers[0]
```

Out[4]:
```
['RHL5', 'RHL6', 'RHL7', 'RHL8']
```

In [5]:
```
servers[0][0]
```

Out[5]:
```
'RHL5'
```

In [6]:
```
1 servers[0][1]
```

Out[6]:
```
'RHL6'
```

In [9]:
```
1 servers[0][-2:]
```

Out[9]:
```
['RHL7', 'RHL8']
```

In [16]:
```
1 servers = [['RHL5','RHL6','RHL7','RHL8'],('Win10','Win11')]
2 print(type(servers))
3 print(type(servers[1]))
4 print(servers[1])
5 servers[1][0]
```

```
<class 'list'>
<class 'tuple'>
('Win10', 'Win11')
```

Out[16]:
```
'Win10'
```

In [17]:
```
1 servers[0][1]='OL6' # we can modify
2 servers
```

Out[17]:
```
[['RHL5', 'OL6', 'RHL7', 'RHL8'], ('Win10', 'Win11')]
```

In [19]:
```
1 servers[1][1]="DEB"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-19-554ae3bb846c> in <module>
----> 1 servers[1][1]="DEB"

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

In [20]:
```
1 servers
```

Out[20]:
```
[['RHL5', 'OL6', 'RHL7', 'RHL8'], ('Win10', 'Win11')]
```

In [26]:
```
1 # list of dict
2 web_apps = [{'url':'https://www.python.org'},{'url':'https://perl.org'},
3             {'url':'https://ruby.org'}]
4
5 print(type(web_apps),len(web_apps))
6 print(type(web_apps[0]))
7 web_apps[0]['url'] # variable[index]['string'] <== list of dict
```

```
<class 'list'> 3
<class 'dict'>
```

Out[26]:
```
'https://www.python.org'
```

In [27]:
```
print(web_apps[0]['url'],web_apps[1]['url'],web_apps[2]['url'])
```

```
https://www.python.org (https://www.python.org) https://perl.org (https://perl.
org) https://ruby.org (https://ruby.org)
```

In [28]:
```
L=[]
L.append("D1")
L.append(13.4)
L.append(['D1','D2','D3'])
L.append(('T1','T2'))
L.append({'K1':'url','K2':'filename'})
L
```

Out[28]:
```
['D1', 13.4, ['D1', 'D2', 'D3'], ('T1', 'T2'), {'K1': 'url', 'K2': 'filename'}]
```

In [32]:
```
fsinfo=[{'fstype':['xfs','ext4','btrfs']},{'fstype':['zfs','ocfs']}]
print(type(fsinfo))
print(type(fsinfo[0]))
fsinfo[0].keys()
```

```
<class 'list'>
<class 'dict'>
```

Out[32]:
```
dict_keys(['fstype'])
```

In [33]:
```
fsinfo[0]['fstype']
```

Out[33]:
```
['xfs', 'ext4', 'btrfs']
```

In [34]:
```
fsinfo[0]['fstype'][1]
```

Out[34]:
```
'ext4'
```

In [35]:
```
import pprint
pprint.pprint(fsinfo)
```

```
[{'fstype': ['xfs', 'ext4', 'btrfs']}, {'fstype': ['zfs', 'ocfs']}]
```

In [43]:
```
d={'K1':'V1','K2':['V2',{'Kx':'Vx','Ky':'Vy','Kz':[1,2,[3,4],5,('T1','T2')],
print(d)
```

```
{'K1': 'V1', 'K2': ['V2', {'Kx': 'Vx', 'Ky': 'Vy', 'Kz': [1, 2, [3, 4], 5, ('T
1', 'T2')], 'K3': ['V1', 'V2', 'V3']}]}
```

In [44]:
```
pprint.pprint(d)
```

```
{'K1': 'V1',
 'K2': ['V2',
        {'K3': ['V1', 'V2', 'V3'],
         'Kx': 'Vx',
         'Ky': 'Vy',
         'Kz': [1, 2, [3, 4], 5, ('T1', 'T2')]}]}
```

In [45]:
```
d['K2'][1]['K3'][1]
```

Out[45]: 'V2'

In [47]:
```
T=([],('D1','D2'),{'K1':'V1','K2':'V2','K3':'V3'})
print(type(T),type(T[0]))
T[0].append('D1')
T[0].append('D2')
T[0].append('D3')
T[0].append('D4')
T
```

```
<class 'tuple'> <class 'list'>
```

Out[47]: (['D1', 'D2', 'D3', 'D4'], ('D1', 'D2'), {'K1': 'V1', 'K2': 'V2', 'K3': 'V3'})

In [49]:
```
T[-1]['K2']
```

Out[49]: 'V2'

In [51]:
```
T[-1]['K2']
```

Out[51]: 'V2'

In [53]:
```
emp={'ename':[],'edepts':[]} # MD 1Key =>more than one values
```

In [55]:
```
emp['ename'].append('Mr.AB')
emp['edepts'].append('sales')
emp
```

Out[55]: {'ename': ['Mr.AB', 'Mr.AB'], 'edepts': ['sales', 'sales']}

In [56]:
```
while(1):
    name = input('Enter an emp name:')
    if(name):
        emp['ename'].append(name)
    else:
        print('Sorry emp name is missed')
    dept = input('Enter a dept:')
    if(dept):
        emp['edepts'].append(dept)
    else:
        print('Sorry emp dept is missed')
    choice=input('Wish to exit press Yes (or) yes ')
    if(choice == 'Yes' or choice == 'yes'):
        break

```

```
Enter an emp name:Tom
Enter a dept:HR
Wish to exit press Yes (or) yes No
Enter an emp name:bibu
Enter a dept:prod
Wish to exit press Yes (or) yes no
Enter an emp name:leo
Enter a dept:prod
Wish to exit press Yes (or) yes n
Enter an emp name:raj
Enter a dept:sales
Wish to exit press Yes (or) yes Yes
```

In [58]:
```
pprint.pprint(emp)
```

```
{'edepts': ['sales', 'sales', 'HR', 'prod', 'prod', 'sales'],
 'ename': ['Mr.AB', 'Mr.AB', 'Tom', 'bibu', 'leo', 'raj']}
```

In [59]:
```
db={'class':'oracle','class':'mysql','class':'sqlite3','class':'sql'}
db
```

Out[59]:
```
{'class': 'sql'}
```

In [60]:
```
db={'K1':{'class':'oracle'},'K2':{'class':'mysql'},
    'K3':{'class':'sqlite3'},
    'K4':{'class':'sql'}
}
pprint.pprint(db)
```

```
{'K1': {'class': 'oracle'},
 'K2': {'class': 'mysql'},
 'K3': {'class': 'sqlite3'},
 'K4': {'class': 'sql'}}
```

In [61]:
```
db['K3']['class']='influxdb' # modification
pprint.pprint(db)
```

```
{'K1': {'class': 'oracle'},
 'K2': {'class': 'mysql'},
 'K3': {'class': 'influxdb'},
 'K4': {'class': 'sql'}}
```

In [62]:
```
print(db['K3']['class'])
```

```
influxdb
```

In [ ]:
```
#Given List
L=["type=ethernet","interface=eth0","onboot=yes"]

|
create an empty dict
iterate a list - use for loop
using split() split single data into multiple value based on =
add list of 0th index - as a key
            1st index - as a value

d={'type':'ethernet','interface':'eth0','onboot':'yes'}
```

In [65]:
```
L=["type=ethernet","interface=eth0","onboot=yes"]

d={}

for v in L:
    La = v.split("=")#print(v)
    d[La[0]]=L[1]
d
```

Out[65]:
```
{'type': 'interface=eth0',
 'interface': 'interface=eth0',
 'onboot': 'interface=eth0'}
```

In [68]:
```
d['interface']='eth1'
d
```

Out[68]:
```
{'type': 'interface=eth0', 'interface': 'eth1', 'onboot': 'interface=eth0'}
```

In [ ]:

```
File Handling
-----------------      input()               print()
Keyboard(<STDIN>)---------->-------Python----------->------Monitor(STDOUT)
                                    | read/write
                                    |
                                FileHandling(Storage)

1. Read data from <FILE> -- python -- display to monitor
2. python -- create a newFile and write data to FILE
3. Read data from <oneFILE>->python->create and write data to anotherFile

open a file      => fileobject= open(filename,"mode") mode - read 'r' write
read a content   => fileobject.read() ->str / fileobject.readlines() ->[]
close a file     => fileobject.close()

create a newfile  =>  fileobj=open("New_resultfile","w")
write a data to file => fileobj.write("SingleString\n")
close a file         => fileobj.close()

```

In [71]:

```
open("IP1.txt",'r')
```

Out[71]:

```
<_io.TextIOWrapper name='IP1.txt' mode='r' encoding='cp1252'>
```

In [72]:

```
fobj = open("IP1.txt",'r')
fobj.read()
```

Out[72]:

```
'list of available services\nhttpd\natd\ncrond\napache2'
```

In [73]:

```
fobj.read()
```

Out[73]:

```
''
```

In [74]:

```
fobj = open("IP1.txt",'r') # open an existing file
s = fobj.read() # read a file content
fobj.close()
print(s) # display file content to monitor
```

```
list of available services
httpd
atd
crond
apache2
```

In [75]:

```
fobj = open("IP1.txt",'r') # open an existing file
fobj.readlines()
```

Out[75]:

```
['list of available services\n', 'httpd\n', 'atd\n', 'crond\n', 'apache2']
```

In [76]:
```
fobj = open("IP1.txt",'r') # open an existing file
L = fobj.readlines()
fobj.close()
print(L)
```

```
['list of available services\n', 'httpd\n', 'atd\n', 'crond\n', 'apache2']
```

In [78]:
```
L[-3:] # last 3 line contents
```

Out[78]:
```
['atd\n', 'crond\n', 'apache2']
```

In [79]:
```
# python ->create a newfile then write data to file
cost=342.52
wobj = open("r1.log","w")
wobj.write("data1\n")
wobj.write("1235\n")
wobj.write("2343.21\n")
wobj.write(str(cost)+"\n")
wobj.write("101,raj,sales,pune,1000\n")
wobj.close()
```

In [80]:
```
# Read a data from <oneFile> - to python - create/write data to anotherFile
fobj = open('r1.log','r')
s = fobj.read()
fobj.close()

wobj = open('r2.log','w')
wobj.write(s)
wobj.close()
```

In [ ]:
```
# create an emp.csv file
# eid,ename,edept,eplace,ecost
# 101,raj,sales,pune,1000
# ...
# ..

|
read an emp.csv file - line by line - readlines()
...
read a dept name from <STDIN>
search/display matched dept details to monitor.

```

In [82]:

```
wobj = open('e1.csv','a')
c=0
while(c < 5):
    empid = input('Enter emp id:')
    empName = input('Enter an emp name:')
    empDept = input('Enter a dept:')
    empCost = input('Enter a emp Cost:')
    wobj.write("{},{},{},{}\n".format(empid,empName,empDept,empCost))
    c=c+1
wobj.close()
```

```
Enter emp id:101
Enter an emp name:raj
Enter a dept:sales
Enter a emp Cost:1000
Enter emp id:102
Enter an emp name:leo
Enter a dept:prod
Enter a emp Cost:2000
Enter emp id:103
Enter an emp name:paul
Enter a dept:prod
Enter a emp Cost:3000
Enter emp id:104
Enter an emp name:xen
Enter a dept:DBA
Enter a emp Cost:4000
Enter emp id:105
Enter an emp name:anu
Enter a dept:HR
Enter a emp Cost:5000
```

In [83]:

```
fobj = open('e1.csv','r')
s = fobj.read()
fobj.close()
print(s)
```

```
101,raj,sales,1000
102,leo,prod,2000
103,paul,prod,3000
104,xen,DBA,4000
105,anu,HR,5000
```

In [91]:

```
fobj = open('e1.csv','r')
L = fobj.readlines()
fobj.close()

dept = input('Enter your dept:')

for v in L:
    if(dept in v):
        print(v.strip()) # remove \n chars at the end.
```

```
Enter your dept:prod
102,leo,prod,2000
103,paul,prod,3000
```

In [ ]:

```
Q1.

Given List
L = ["type=ethernet","interface=eth0","onboot=yes",
    "bootproto=dhcp","defroute=yes"]

Write a python program
step 1: create a newfile (network.conf)

step 2: iterate list ->split each string into multiple value based on =
step 3: write splitted data into external file(network.conf) in INI format
        ex: type=ethernet
            interface=eth0
##########################################################################
```

In [93]:

```
L = ["type=ethernet","interface=eth0","onboot=yes","bootproto=dhcp","defrout

wobj = open('network.conf','w')
for v in L:
    v1,v2 = v.split("=")
    wobj.write("{}={}\n".format(v1,v2))
wobj.close()
```

In [ ]:

```
Q2.
Write a python program:
step 1: create an empty dict
step 2: read a network.conf file - line by line
        split each line into multiple values
        initialize splitted values to dictionary (d[Key]=Value)
step 3: using for loop - display key - value

step 4: dictionary operation - modify interface eth0 =>eth1
                               modify bootproto dhcp =>static
                               add newIPAddress   IPADDR = 10.20.30.40
                               add  prefix value  prefix = 24
step 5: display updated dict details(Key - Value) - step 3

step 6: create a newConfig (network1.conf) file, write updated dict
        content to external(network1.conf) file in INI format.

```

In [102]:

```
d={} # empty dict

with open('network.conf','r') as fobj:
    L = fobj.readlines()
    for v in L: # read a file content line by line
        v=v.strip() # remove \n
        K,V=v.split("=") # split each line into multiple values
        d[K]=V # add splitted data into dict

print("Key \t  Value:")
for v in d:
    print("{}\t{}".format(v,d[v]))

d['interface']='eth1'    # dict - modification
d['bootproto']='static'  # dict - modification
d['IPADDR']='10.20.30.40' # add new item into an existing dict
d['PREFIX']=24 # add new item into an existing dict

print("\nUpdated dict details:-\n")
print("Key \t  Value:")
for v in d:
    print("{}\t{}".format(v,d[v]))


with open("network1.conf","w") as wobj:
    for v in d:
        wobj.write("{}={}\n".format(v,d[v]))
```

```
Key       Value:
type    ethernet
interface       eth0
onboot  yes
bootproto       dhcp
defroute        yes

Updated dict details:-

Key       Value:
type    ethernet
interface       eth1
onboot  yes
bootproto       static
defroute        yes
IPADDR  10.20.30.40
PREFIX  24
```

In [103]:

```
def fx():  # function definition
    print('This is fx code block')
```

In [105]:

```
fx() # simple function call
```

```
This is fx code block
```

In [106]:

```
def fx(a):
    print("a={}".format(a))
    print(type(a))

fx(10)
fx(10.0)
fx('data')
fx(b'data')
fx([])
fx(('d1','d2'))
fx({'K1':'Va'})
fx({'K1','K2'})
```

```
a=10
<class 'int'>
a=10.0
<class 'float'>
a=data
<class 'str'>
a=b'data'
<class 'bytes'>
a=[]
<class 'list'>
a=('d1', 'd2')
<class 'tuple'>
a={'K1': 'Va'}
<class 'dict'>
a={'K2', 'K1'}
<class 'set'>
```

In [108]:

```
# fx() TypeError: fx() missing 1 required positional argument: 'a'
# fx(1,2) TypeError: fx() takes 1 positional argument but 2 were given

```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-108-a0f28155e967> in <module>
      1 # fx() TypeError: fx() missing 1 required positional argument: 'a'
----> 2 fx(1,2)

TypeError: fx() takes 1 positional argument but 2 were given
```

In [111]:
```
L=[]
L.append(10,20,30,40)
help(list.append)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-111-e3859a2b2181> in <module>
      1 L=[]
----> 2 L.append(10,20,30,40)
      3 help(list.append)

TypeError: append() takes exactly one argument (4 given)
```

In [115]:
```
def login(uname="root",passwd="Welcome"): # default args
    print("user name={} password={}".format(uname,passwd))

login()
login('admin')
login('admin','ABC-XYZ')
login('admin','ABC','1234')
```

```
user name=root password=Welcome
user name=admin password=Welcome
user name=admin password=ABC-XYZ

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-115-aeb067404206> in <module>
      5 login('admin')
      6 login('admin','ABC-XYZ')
----> 7 login('admin','ABC','1234')

TypeError: login() takes from 0 to 2 positional arguments but 3 were given
```

In [116]:
```
def display():
    global port
    port=5000

display()
```

In [118]:
```
print("port value is =",port)
```

```
port value is = 5000
```

In [120]:
```
def display():
    return 10
display()
```

Out[120]:
```
10
```

In [121]:
```
rv = display()
print(rv)
```

```
10
```

In [124]:
```
def fx():
    print('this is fx block')
    return 15

rv=fx()
print(rv)
```

```
this is fx block
15
```

In [125]:
```
def display():
    return "STDOUT","STDERR"

display()
```

Out[125]:
```
('STDOUT', 'STDERR')
```

In [126]:
```
def fx():
    pass

if(False):
    pass
else:
    print('Thank you')
```

```
Thank you
```

In [ ]:

```
Module
-----------
|__ existing python file
|__ reusability  =>  import <module>
                     <module>.<member>
                                  |__variable,function,class etc.,


|__ file: D:\> sab.py            D:\> p1.py           D:\> p2.py
    ------------------           -----------------    -------------
    def  fx():                    import sab          import sab
        '''                       print(sab.fx())     print(sab.myvar)
        fx operation              -------------------- -------------
        '''                       D:\>python p1.py    D:\>python p2.py
        print('fx-value')        fx-value             120

    myvar=120
    --------------------
    D:\> python sab.py{Enter}
    D:\>

    import <module>
    <module>.<member>
           Vs
    from <module> import <member>
    <member>

    E:\> p3.py
    ----------------
    import sab
    sab.fx()
    -----------------
    E:\> python p3.py
    Module Not Found - there is no module sab

    file: http_template_code.py

    import http_template_code   =>    import http_template_code as hc
    http_template_code.<member>       hc.<member>

    import numpy as np
    import pandas as pd
```

In [127]:

```
s="abc"
s.upper()
```

Out[127]: 'ABC'

In [128]:

```
s
```

Out[128]: 'abc'

In [129]:
```
L=[]
L.append("D1")
```

In [130]:
```
L
```

Out[130]:
```
['D1']
```

In [131]:
```
myL=[]

def fx(a):
    if(type(a) is list):
        a.append("D1")
print(myL)
```

```
[]
```

In [132]:
```
fx(myL)
```

In [133]:
```
print(myL)
```

```
['D1']
```

In [134]:
```
import copy
L1=copy.copy(myL) # shallow copy
```

In [135]:
```
L1.append("D2")
L1.append("D3")
L1
```

Out[135]:
```
['D1', 'D2', 'D3']
```

In [136]:
```
myL
```

Out[136]:
```
['D1']
```

In [ ]:

```
int,float => numerical initialization ; arithmetic
---------
s='456' =>int(s)
v=456 -> str(v) ->'456'

type(variable) =><class 'int/float/str/bytes/list...'>

s1="A"
s2="a"

in not in <== membership  Vs  == relational operator    Vs  is is not
---------                      |_character by character       memory/object
search a pattern                      value

break -exit from loop ; not exit from script
return - exit from function block ; not exit from script

list ,tuple,dict, set , str
        |               |
      immutable        immutable

list - collection of ordered items - mutable - []  <== Variable[index]
tuple - collection of ordered items -immutable - () <== Variable[index]
dict  - collection of unordered items - mutable - {} <== Variable['Key']
 |__associated array;hash
set  - collection of unordered items - not index,key:value
 |__Not allows duplicate items

import copy
copy.copy()  Vs copy.deepcopy()
      |               ----
     shallow

union()        |
intersection() &
difference()   -
symmetric_difference()  ^

del(Listname[index]) ->None   Vs Listname.pop() ->removed_lastindex
del(dictname[key]) ->None    Vs dictname.pop(key) ->removed_Value

open(filename,mode)
               |__r - read; w - write; a - append
                           |          |_won't overwrite if file is already
                           |__file is already exists - overwrite

    open(filename,"r") same as open(filename)
    open(filename,"w") <== writing

def fx(a1,a2,a3,a4=True,a5=0.0,a6=1,*args,**kwargs)
                                      |__tuple |___dict
        global var
        var=10
var variable is accessible globally

global
```

```
return
--------//function keywords - we can use inside the function only

import <module>     Vs    from <module> import <member>
<module>.<member>          <member>

import <module> as <userdefinedName>
..
import sys
import os
import csv
import json

pass - empty  code block
Vs
None - NoneType
''   - empty string (str type) -
```