Python veriable

In [1]:
```
# x is veriable which store numbers.
x=10
y=20
print(x+y)
```

```
30
```

Comparison operator <>,==,!=

In [2]:
```
a=10
b=30
print(a==b)
print(a!=b)
print(a>b)
print(a<b)
```

```
False
True
False
True
```

logical operator and,or,not

In [3]:
```
number=22
number2=33
number3=44
print(number>number3 and number>number2)
print(number<number3 or number>number2)
print (not(number<number3 and number<number2))# reverse order use :not:
```

```
False
True
False
```

Membership operator in ,not in

In [4]:
```
name='HASSAN'
print('s' in name)
print("S" in name)
```

```
print("S"  not in name)
print("P" in name)
```

```
False
True
False
False
```

In [5]:
```
l=(1,2,3,4,5,5)
print(type(l))
l=[1,2,3,4,5,5]
print(type(l))
l={1,2,3,4,5,5}
print(type(l))
2 in l
```

```
<class 'tuple'>
<class 'list'>
<class 'set'>
```
Out[5]:
```
True
```

identity operator , is ,is not

In [6]:
```
x=20
y=30
print(x is y)
x is not y
```

```
False
```
Out[6]:
```
True
```

bitwise operator OR AND ,XOR

In [7]:
```
a=10
b=30
c=40
print(a>b | a>c)
print(a<b & a<c)
print(bin(a))
```

```
False
False
0b1010
```

Data type (tuple ,list,set,dictionary,int float ,complex,string,byte array)

In [8]:
```
s="hellow world"
print(s,type(s))
```

```
hellow world <class 'str'>
```

In [9]:
```
# List,[],immutable list
a=[2,3,5,5.5,6,3]
print(a,type(a))
a.append(22)
print(a)
a.remove(2)
print(a)
```

```
[2, 3, 5, 5.5, 6, 3] <class 'list'>
[2, 3, 5, 5.5, 6, 3, 22]
[3, 5, 5.5, 6, 3, 22]
```

list and dictionary or mutable but tuple and set or immutable D-type

In [10]:
```
##############################oprations in
list(append,extend,insert,remove,pop,slice
,reverse,min,max,count)############


my_list=[1,2,3,4,5,5]
print(type(my_list))
# append function simply add number in end of list item.
print("Äppend")
my_list.append(88)
print(my_list)
print("extend")
# extend functipon increasing the list size .
for i in range(10):


    my_list.extend([i])
print(my_list)
print("insert")
```

```
# insert functipon used  two value:first for position and sercond is value.
my_list.insert(2,22)
print(my_list)

# remove functipon used for delete item in list.
print("Remove functipon")
my_list.remove(22)
print(my_list)

print("slice function")
#Slice functipon used for show value in limit.
print(my_list[2:5])
print(my_list[3:])
print(my_list[:3])
```

```
<class 'list'>
Äppend
[1, 2, 3, 4, 5, 5, 88]
extend
[1, 2, 3, 4, 5, 5, 88, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
insert
[1, 2, 22, 3, 4, 5, 5, 88, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Remove functipon
[1, 2, 3, 4, 5, 5, 88, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
slice function
[3, 4, 5]
[4, 5, 5, 88, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 2, 3]
```

In [11]:

```
# dictionary oprations,
(pop,get,update,copy,keys,values,formkeys)##############################
d=dict({
    'name': 'Hassan',
    'roll': 21,
    'result':[1,2,3,4,5]
})
print(d)
print(type(d))
# add item in dictionary:
print("add")
d['class']=12
print(d)
```

```
print("get")
gt=d.get('roll')
print(gt)
print("copy")
d.copy()

d.update({'color':'black'})
print(d)

d.keys()
d.values()
# formkeys are used to create the dictionary which have same
values#######################################
x=('k1','k2','k3','k4','k5','k6','k7')
y=('0')
xy=dict.fromkeys(x,y)
print(xy)
```

```
{'name': 'Hassan', 'roll': 21, 'result': [1, 2, 3, 4, 5]}
<class 'dict'>
add
{'name': 'Hassan', 'roll': 21, 'result': [1, 2, 3, 4, 5], 'class': 12}
get
21
copy
{'name': 'Hassan', 'roll': 21, 'result': [1, 2, 3, 4, 5], 'class': 12, 'color': 'black'}
{'k1': '0', 'k2': '0', 'k3': '0', 'k4': '0', 'k5': '0', 'k6': '0', 'k7': '0'}
```

In [12]:

```
########################## Nested
dictionary:##########################################
classes=dict({
            'nine_class':
            {
                'name':'raza',
                'age ':'22',
                'blood group':'B+'
            },

            'ten_class':
            {
```

```
            'arts_group':

                    {
                    'name':'raza',
                    'age':22,
                    'blood group':'B+'
                    },
            'science_group':
                    {
                    'name':['raza','ali','hassan'],
                    'age': 22,
                    'blood group':'B+'
                    }
            }
})
print(classes)
print(classes['ten_class']['science_group']['name'])
print(classes['ten_class']['science_group']['age'])
```

```
{'nine_class': {'name': 'raza', 'age ': '22', 'blood group': 'B+'}, 'ten_class': {'arts_
group': {'name': 'raza', 'age': 22, 'blood group': 'B+'}, 'science_group': {'name': ['ra
za', 'ali', 'hassan'], 'age': 22, 'blood group': 'B+'}}}
['raza', 'ali', 'hassan']
22
```

In [13]:

```
# tupe opeartaion (length,type)
a=(1,2,3,4,5)
print(type(a))
print(len(a))
```

```
<class 'tuple'>
5
```

In [14]:

```
a={1,2,3,4,5,6}
print(type(a))
x={1,2,3,4,5.5}
print(x)
print(type(x))
```

```
<class 'set'>
{1, 2, 3, 4, 5.5}
<class 'set'>
```

conditional statment if,if else,elif

In [15]:
```
%load_ext jupyternotify
```

In [16]:
```
%reload_ext jupyternotify
```

In [17]:
```
x=9
import time
time.sleep(3)
%notify -m " this take {x} time"
print("done!")
```

done!

In [18]:
```
# use """ three double qotations for writting multiple lines in python.

counter=1;
while counter<=2:
    print("""Press "+" for addition
    Press "-" for subtraction
    Press "/" for division
    Press "*" for multiplication""")

    num1=int(input("enter the first number"))
    num2=int(input("enter the secound number"))
    operation=input("enter the operation:")
    if operation=='+':

        print("result=",num1+num2)
    elif operation=='-':

        print("result=",num1-num2)
    elif operation=='*':

        print("result=",num1*num2)
    elif operation=='/':
```

```
        print("result=",num1/num2)
    else:
        print("Invalid operation")
    counter +=1
else:

    import time
    time.sleep(3)
    %notify -m f"Finished! No longer then time {counter} "
    print('Finished!')
```

```
Press "+" for addition
    Press "-" for subtraction
    Press "/" for division
    Press "*" for multiplication
enter the first number1
enter the secound number2
enter the operation:+
result= 3
Press "+" for addition
    Press "-" for subtraction
    Press "/" for division
    Press "*" for multiplication
enter the first number2
enter the secound number3
enter the operation:/
result= 0.6666666666666666

Finished!
```

loop for,while

In [19]:

```
for i in range(5):
    print((i))
# strat and end limit write in range function.
for i in range(1,5):
    print(i)
```

```
0
1
2
3
4
```

```
1
2
3
4
```

In [20]:
```
# start,end,and (gap) is given in end
for i in range(1,10,2):
    print(i)
```

```
1
3
5
7
9
```

In [21]:
```
# while loop
i=0
while i<=5:
    print("python")
    i+=1
```

```
python
python
python
python
python
python
```

indexing and slicing

In [22]:
```
string="pyt hon"
print(string)
```

```
pyt hon
```

In [23]:
```
print(string[3::4])
```

In [24]:
```
print(string[0:7])
print(type(string))
print(string[0])
```

```
pyt hon
<class 'str'>
p
```

In [25]:
```
for i in range(len(string)):
    print(string[i],[i])
if i==0:
    print(string[i])
print(string[3])
```

```
p [0]
y [1]
t [2]
  [3]
h [4]
o [5]
n [6]
```

upper,lower,captilize,title

In [26]:
```
%%javascript
name="Ha"
print("name")
```

In [27]:
```
list1=[1,2,3,4,5,6]
length=len(list1)
length
```

Out[27]:
```
6
```

In [28]:
```
%load_ext jupyternotify
```

```
The jupyternotify extension is already loaded. To reload it, use:
  %reload_ext jupyternotify
```

In [29]:
```
for i in range(length-1,-1,-1):
    # firts (-1) used for limit start and second(-1) used for limit end and third used for gap.*ValueError: range() arg 3 must not be zero
    print(list1[i])
```

```
6
5
4
3
2
1
```

In [30]:
```
```

```
list1=[1,2,3,4,5,6]
list2=[88,99,101]
print(type(list1))
list1.insert(2,99)
print(list1)
list1.extend(list2)
list1
```

```
<class 'list'>
[1, 2, 99, 3, 4, 5, 6]
```

Out[30]:
```
[1, 2, 99, 3, 4, 5, 6, 88, 99, 101]
```

list comprehension

In [31]:
```
# syntax(expression,condition)
list_cmp=[a*3 for a in range(1,10) if a%2!=0 ]
print(list_cmp)


for i in range(1,10):
     print(f"{i}*2=",i*2)
```

```
[3, 9, 15, 21, 27]
1*2= 2
2*2= 4
3*2= 6
4*2= 8
5*2= 10
6*2= 12
7*2= 14
8*2= 16
9*2= 18
```

stack and queu in python list

In [32]:
```
%load_ext jupyternotify
```

```
The jupyternotify extension is already loaded. To reload it, use:
  %reload_ext jupyternotify
```

In [33]:
```
# stack(push,pop,peek,display) FILO(first in last out)
new_l=[]
while True:
    i=int(input("""
```

```
1 for push
2 for pop
3 for display
4 for peek
5 for break
"""))

if i==1:
    n=int(input("enter the element:"))
    new_l.append(n)
    print(new_l)
elif i==2:
    if len(new_l)==0:
        print("empty")
    else:
        new_l.pop()
        print(new_l)
elif i==3:
    if len(new_l)==0:
        print("empty")
    else:
        print(new_l)
elif i==4:
    if len(new_l)==0:
        print("empty")
    else:
            print("peek list",new_l[-1])
            print(new_l)
elif i==5:

    import time
    time.sleep(3)
    %notify -m(completed)
    print("completed")
    break;

else:
    print("invalid command ")
```

```
    1 for push
    2 for pop
    3 for display
    4 for peek
    5 for break
    1
enter the element:20
[20]

    1 for push
    2 for pop
    3 for display
    4 for peek
    5 for break
    1
enter the element:40
[20, 40]

    1 for push
    2 for pop
    3 for display
    4 for peek
    5 for break
    2
[20]

    1 for push
    2 for pop
    3 for display
    4 for peek
    5 for break
    4
peek list 20
[20]

    1 for push
    2 for pop
    3 for display
    4 for peek
    5 for break
    5

completed
```

queu [FIFO]

In [34]:

```
# stack(push,pop,peek,display) FILO(first in last out)
new_l=[]
while True:
    i=int(input("""
```

```
1 for enque
2 for deque
3 for display
4 for first element
5 for last element
6 for exit
"""))

if i==1:
    n=0
    n1=int(input("enter the first limit:"))
    n2=int(input("enter the last limit:"))
    x=[new_l for new_l in range(n1,n2)]
    new_l.append(x)
    print(new_l)

elif i==2:
    if len(new_l)==0:
        print("empty")
    else:
        del new_l[0]
        print(new_l)
elif i==3:
    if len(new_l)==0:
        print("empty")
    else:
        print(new_l)
elif i==4:
    if len(new_l)==0:
        print("empty")
    else:
            print("peek list",new_l[0])
            print(new_l)
elif i==5:
    if len(new_l)==0:
            print("empty")
    else:
            print("peek list",new_l[-1])
            print(new_l)
```

```
    elif i==6:
        import time
        time.sleep(3)
        %notify -m(completed)
        print("completed")
        break;

    else:
        print("invalid command ")
```

```
    1 for enque
    2 for deque
    3 for display
    4 for first element
    5 for last element
    6 for exit
    1
enter the first limit:20
enter the last limit:80
[[20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 4
1, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 6
3, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79]]

    1 for enque
    2 for deque
    3 for display
    4 for first element
    5 for last element
    6 for exit
    2
[]

    1 for enque
    2 for deque
    3 for display
    4 for first element
    5 for last element
    6 for exit
    3
empty

    1 for enque
    2 for deque
    3 for display
    4 for first element
    5 for last element
    6 for exit
    4
```

```
empty

    1 for enque
    2 for deque
    3 for display
    4 for first element
    5 for last element
    6 for exit
    6

completed
```

dictionay-function

In [35]:

```
# del,update,get,pop,clear,keys
d={
    'college':{# college
        'name':'zamindar',
        'location':'gujrat',
        'classes':{# college>classes
         'block1':{#college>classes>block1
             'name':'ali',
             'location':'left',
             'color':'green'

         },
          'block2':{#college>classes>block1
              'name':'qaid',
             'location':'right-left',
             'color':'white',
              'area':200*500
          }


        }
    }
}
print(d['college']['classes']['block2']['color'])
d.keys()
d.copy()
x=d.update({
    'university':'123'
```

```
})
print(x)
del d['college']['classes']['block2']['color']
print(d)
```

```
white
None
{'college': {'name': 'zamindar', 'location': 'gujrat', 'classes': {'block1': {'name': 'a
li', 'location': 'left', 'color': 'green'}, 'block2': {'name': 'qaid', 'location': 'righ
t-left', 'area': 100000}}}, 'university': '123'}
```

user define function

simple function,

function with argument,

return type function,

In [36]:
```
#simple function
def name():
    print("hassan")
name()
# argumented value
def sum(a,b):
    print(a+b)
sum(10,20)

def sum(a,b=30):
    print(a+b)
sum(20)

def mult(x,y):
    result=x*y
    return result
mult(10,20)
```

```
hassan
30
50
```

Out[36]:
```
200
```

math module

In [37]:
```
import math
x=-1.8
y=3
# math,ceil function chnage point value to maximum value
print(math.ceil(x))
#math.floor function change value to minium point value
print(math.floor(x))

print(math.fabs(x))#mathh.fabs function minius value to positive value

#factorial.....
print(math.factorial(y))
#fsum calculate the sum of list and tuple
l=[1,2,3,4,5,6,7,8,9]
print(math.fsum(l))
# to find sqrt
print(math.sqrt(9))
```

```
-1
-2
1.8
6
45.0
3.0
```

In [38]:
```
'''
Euler’s Number
Pi
Tau
Infinity
Not a Numer (NaN)
'''
print(math.e)
print(math.pi)
print(math.tau)
print(math.inf)
print(math.nan)
#gcd() function is used to find the greatest common divisor of two numbers
a=15
```

```
b=9
print(math.gcd(a,b))
#exp() method is used to calculate the power of e i.e. e^y e=(2.718)
exponent
num=4
print(math.exp(num))

print(math.pow(2,9)) # 2 is base and 9 is power.
#logrithm function
print(math.log10(2))
print(math.log(3))
print("trignometry function sin(),cos,tan")
fun=math.pi/3
print(math.cos(fun))
```

```
2.718281828459045
3.141592653589793
6.283185307179586
inf
nan
3
54.598150033144236
512.0
0.3010299956639812
1.0986122886681098
trignometry function sin(),cos,tan
0.5000000000000001
```

In [ ]:

Random module

In [39]:

```
import random
x=random.randint(1,10)
print(x)
#  show value from 0 to 1 in point base
y=random.random()
print(y)

l=[20,40,50,90]
random.shuffle(l)
print(l)
```

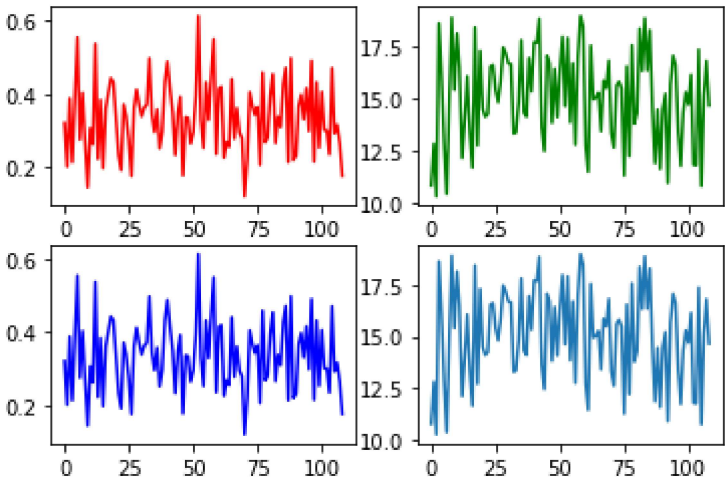
```
z=random.choice(l)
print(z)
# uniform module show value between number in floating form
u=random.uniform(2,4)
print(u)
# seeds
random.seed(5)
print(random.random())
# getstate and setstate in random
#state=random.getstate()
print(random.sample(range(10),k=3))
#random.setstate()
#randrange function --------------
x=random.randrange(1,20,7)#random.randrange(start(opt),stop,step(opt))

print(x)
x=random.randint(1,20)#randint(start, end)
print(x)

low=10
heigh=50
mode=40
tr=random.triangular(low,heigh,mode)
print(tr)
```

```
5
0.28189769530927133
[50, 90, 20, 40]
20
2.1313670576772292
0.6229016948897019
[5, 8, 0]
8
8
37.9064412593994
```

In [40]:

```
# simple plot and subplopt data.
import matplotlib.pyplot as plt
import random
lst=[]
lst2=[]
```

```
for i in range(109):
    x=random.betavariate(10,20)
    y=random.triangular(10,20)

    lst.append(x)
    lst2.append(y)



fig, axs = plt.subplots(nrows=2, ncols=2)
axs[0,0].plot(lst,color='red',)
axs[0,1].plot(lst2,color="green")
axs[1,0].plot(lst,color='blue')
axs[1,1].plot(lst2)



plt.show(lst)
```



date and time

In [ ]:

```
```

In [41]:

```
import datetime
#datetime used for current time
x=datetime.datetime.now()
print(x)
# strftime function used for shoing date and time in full form
sh=x.strftime("%Y,%m,%d,%B")
print(sh)
```

```
2022-01-18 22:01:18.886599
2022,01,18,January
```

In [42]:

```
import random # random module used for pick random value in list set
x=random.randint(1,10)

while True:
        try:
            guess=int(input("enter the guess number:"))
        except ValueError:
            print("Pease enter integer value only")
            continue;
        if guess>x:
            print("your number is greater:")
        elif guess<x:
            print("your number is smaller")
        elif guess==x:
            print(f'{guess}=={x}')
            print("you win this game!")
            break;
```

```
enter the guess number:8
8==8
you win this game!
```

Rock,paper,scissor game

In [43]:

```
import random
l=['rock','paper','scissor']
user=int(input('''
1 Game start...
2 end |Exit

'''))
user_count=0
computer_count=0
while True:
    if user==1:
        for i in range(1,6):
            u_choice=int(input('''
```

```
            1-rock
            2-paper
            3-scissor
            '''))
            if u_choice==1:
                user_choice="rock"

            elif u_choice==2:
                user_choice="paper"
            elif u_choice==3:
                user_choice="scissor"
            computer_choice=random.choice(l)

            if user_choice==computer_choice:
                computer_count=computer_count+1
                user_count =user_count+1

                print("user_choice",user_choice)
                print("computer_choice",computer_choice)

            elif (user_choice=="rock" and computer_choice=="scissor")
or(user_choice=="paper"  and computer_choice=="rock")
or(user_choice=="scissor" and computer_choice=="paper"):

                print("You win.....")
                print("user_choice",user_choice)
                print("computer_choice",computer_choice)
                user_count =user_count+1

            else:

                print("computer win...")
                print("user_choice",user_choice)
                print("computer_choice",computer_choice)
                computer_count=computer_count+1

        if user_count==computer_count:
            print("game over:")
            print("user_count",user_count)
```

```
            print("computer_count",computer_count)
            break
        elif user_count>computer_count:
            print("you win this game:")
            print("user_count",user_count)
            print("computer_count",computer_count)
            break
        else:

            print("computer win this game:")
            print("user_count",user_count)
            print("computer_count",computer_count)
            break

    else:

        break
```

```
1 Game start...
2 end |Exit

1

            1-rock
            2-paper
            3-scissor
            1
user_choice rock
computer_choice rock

            1-rock
            2-paper
            3-scissor
            2
computer win...
user_choice paper
computer_choice scissor

            1-rock
            2-paper
            3-scissor
            3
computer win...
user_choice scissor
computer_choice rock
```

```
            1-rock
            2-paper
            3-scissor
            2
computer win...
user_choice paper
computer_choice scissor

            1-rock
            2-paper
            3-scissor
            3
You win.....
user_choice scissor
computer_choice paper
computer win this game:
user_count 2
computer_count 4
```

pickle method.

In [44]:

```
# What is pickling?
# Pickle is used for serializing and de-serializing Python object
structures, also called marshalling or flattening.
# Serialization refers to the process of converting an object in memory to
a byte stream that can be stored on disk or sent over a network.
# data in Random Access Memory (RAM)) to another (text on disk).
# Your program's state data can be saved to disk, so you can continue
working on it later on.
# What can be pickled?
# You can pickle objects with the following data types:
# '''
# Booleans,
# Integers,
# Floats,
# Complex numbers,
# (normal and Unicode) Strings,
# Tuples,

# Lists,
# Sets, and
# Dictionaries that ontain picklable objects.
```

In [45]:
```
# serlization.pickle dump data.
import pickle
student_data={
    'name':'ali',
    'class':13,
    'age':25,
    'Region':'Lahore',
    'status':'Pass',
}
filename='abc.txt' # create file name
input_data=open('abc.txt','wb')# file in writting mode(wb)
pickle.dump(student_data,input_data) #dump data
input_data.close()
print(type(input_data))
print(input_data)
```

```
<class '_io.BufferedWriter'>
<_io.BufferedWriter name='abc.txt'>
```

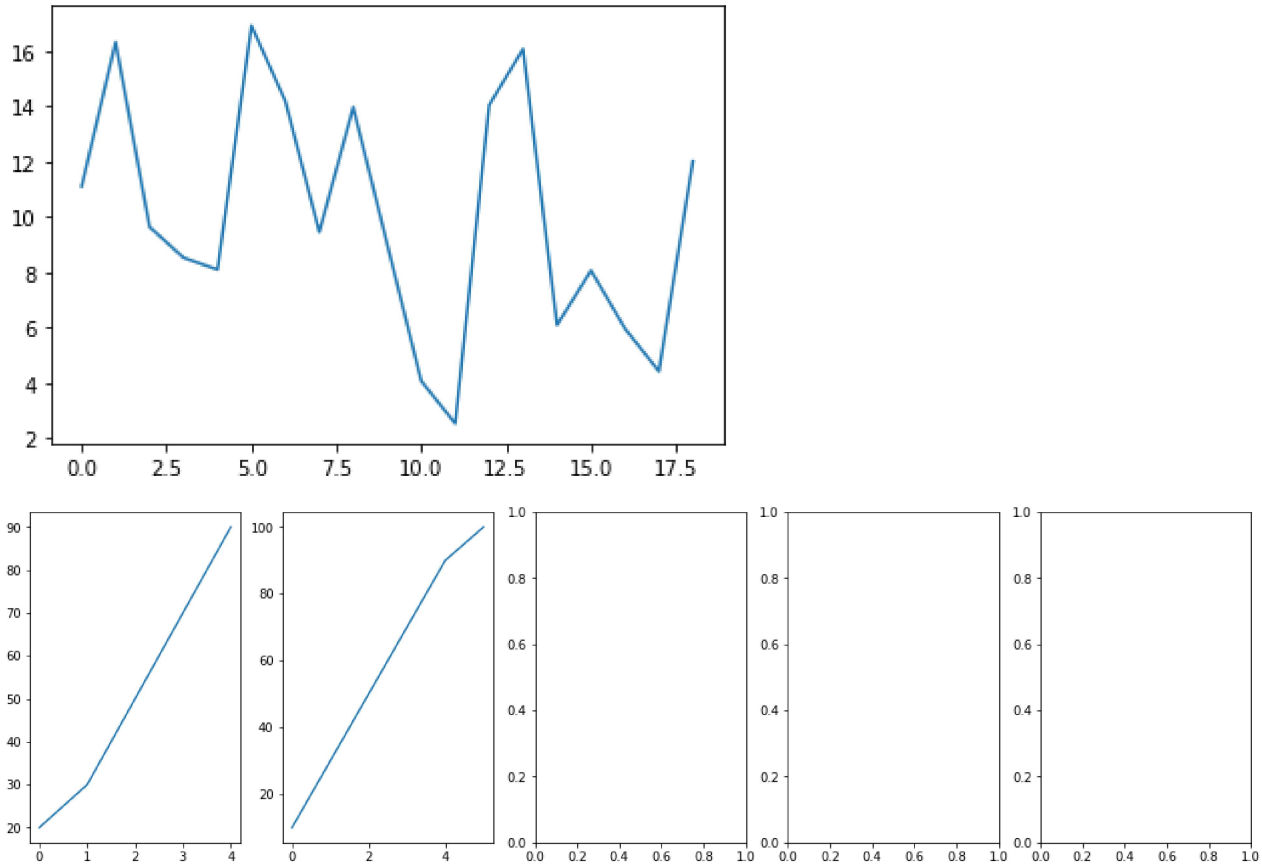
In [46]:
```
# unpickle data/de-serlization/load data
out_data=open(filename,'rb')
x=pickle.load(out_data)
out_data.close()
print(x)
print(type(x))
```

```
{'name': 'ali', 'class': 13, 'age': 25, 'Region': 'Lahore', 'status': 'Pass'}
<class 'dict'>
```

In [47]:
```
import pickle
import pandas as pd
import matplotlib.pyplot as plt
df=pd.read_excel("Book1.xlsx")
print(df.head(0))
plt.plot(df)
plt.show()
import matplotlib.pyplot as plt
import numpy as np
l=np.array([20,30,50,70,90])
y=np.array([10,30,50,70,90,100])
```

```
fig,axs=plt.subplots(nrows=1,ncols=5,figsize=(18, 5))



axs[0].plot(l)
axs[1].plot(y)

plt.show()
```

```
Empty DataFrame
Columns: [7.41403727732135]
Index: []
```



JSON

In [48]:

```
import json
clas = {
    "nine":{
        "name":9,
        'stregth':88,
        'location':"east"
    },
    'ten':{
```

```
         "name":9,
        'stregth':88,
        'location':"east"
    }
}
print(type(clas))
s=json.dumps(clas)
with open('jsn.txt','w') as f:
    f.write(s)

print(s)
print(type(s))



f=open('jsn.txt','r')
s=f.read()
s
z=json.loads(s)
print(type(z))
```

```
<class 'dict'>
{"nine": {"name": 9, "stregth": 88, "location": "east"}, "ten": {"name": 9, "stregth": 8
8, "location": "east"}}
<class 'str'>
<class 'dict'>
```

oop

In [49]:

```
# classes and object
# function/method can create out of classes.but when function made in
classes (self) must be used.
class electronic:
    def computer(self,name,genration,model,processor):
        self.name=name
        self.genration=genration
        self.model=model
        self.processor=processor
        return self.name,model,genration,processor
    def mobile(self,company,memomry,ram):
        self.company=company
```

```
        self.memomry=memomry
        self.ram=ram
        return self.company,memomry,ram
    def __init__(self,screen,camera,battry): # constructor no need of
object.
        self.screen=screen
        self.camera=camera
        self.battry=battry

        print(self.screen,camera,battry)

obj=electronic('HD','6mp','5000mah')
obj.computer('hp','5th','2021','i5')
print(obj.mobile('Oopo',32,3))
def multiple(a,b):
    c=a*b
    return c
multiple(2,7)
```

```
HD 6mp 5000mah
('Oopo', 32, 3)
```

Out[49]: 14

In [50]:

```
class student:
    def __init__(self,name,id,age):
        self.name=name
        self.age=age
        self.id=id

    def display(self):
        print("name detail")
        print(self.name,self.id,self.age)
obj=student('ali','222oor','22')
obj.display()
# class attributes
print(obj.__dict__)
print(obj.__doc__)
print(obj.__class__)
print(obj.__dir__)
print(obj.__str__)
```

```
name detail
ali 222oor 22
{'name': 'ali', 'age': '22', 'id': '222oor'}
None
<class '__main__.student'>
<built-in method __dir__ of student object at 0x0000026C4EF3D460>
<method-wrapper '__str__' of student object at 0x0000026C4EF3D460>
```

constructor&destructor in python

In [51]:
```
%load_ext jupyternotify
```

```
The jupyternotify extension is already loaded. To reload it, use:
  %reload_ext jupyternotify
```

In [52]:
```
# Destructors are called when an object gets destroyed. In Python,
destructors are not needed as much needed in C++
# because Python has a garbage collector that handles memory management
automatically.
import datetime
try:
    class auto:
        def __init__(self):
            print("first item is car")
        def __del__(self):# def __del__() used for destroying item;
            print("item destroyed")
        def car(self):
            print("car color is white")
    obj=auto()
    del obj # now object is get destroyed and not able to used again for
data.


    obj.car()
except NameError:
    import time
    time.sleep(3)
    %notify -m(obj-not-defined)
    print("obj is destroyed by destructor")
```

```
first item is car
```

```
item destroyed

obj is destroyed by destructor
```

Inheritance:

In [53]:
```
# simple inheritance
# multilevel inheritance(one class inherid in second class)
# multiple inheritance
class car:
    def feature_car(self,name,engine,average):
        self.name=name
        self.engine=engine
        self.average=average
        return self.average,engine,name
class tractor(car):
    def feature_tractor(self,color,number):
        self.color=color
        self.number=number
        return self.color,number
class van():
    def feature_van(self,licence):
        self.licence=licence
        return self.licence


obj=tractor() # we create chilled class object and inherid properties from
parent class:
print(obj.feature_tractor('red','12234'))
print(obj.feature_car('honda','3.2','66'))
```

```
('red', '12234')
('66', '3.2', 'honda')
```

In [54]:
```
# multilevel inheritance(one class inherid in second class and second in
third classs)grand-parent->parent->chilled


class vahical:
    def feature(self,name,color,average):
        self.name=name
        self.color=color
```

```
        self.average=average
        return self.average,color,name
class tractor(vahical):
    def feature_tractor(self,engine,number):
        self.engine=engine
        self.number=number
        return self.engine,number
class van(tractor):
    def feature_van(self,licence):
        self.licence=licence
        return self.licence

obj=van() # we create chilled class object and inherid properties from
parent class:
print(obj.feature_tractor('red','12234'))
print(obj.feature('honda','red','66'))
```

```
('red', '12234')
('66', 'red', 'honda')
```

In [55]:

```
# multiple inheritance  two parents->one chilled
class vahical:

    def __init__(self):
        print("vahical is parent class")
    def feature(self,name,color,average):
        self.name=name
        self.color=color
        self.average=average
        return self.average,color,name

class tractor():

    def feature_tractor(self,engine,number):
        self.engine=engine
        self.number=number
        return self.engine,number
    def tr(self):
        print("tHIS IS my tractor")
```

```
class van(vahical,tractor): # show multilevel inheritance :
    def feature_van(self,licence):
        self.licence=licence
        return self.licence
        super().feature
        super().feature_tractor
        super().tr

        print("derived")

obj_van=van() # we create chilled class object and inherid properties from
parent class:
obj_van.feature_van("ok")

# print(obj_van.feature_tractor('red','12234'))
# print(obj_van.feature('honda','red','66'))
```

```
vahical is parent class
'ok'
```

Out[55]:

In [56]:

```
class A:
    def __init__(self):
        print(" This is class A")
class B:
    def __init__(self):
        print(" This is class B")
class C(A,B): # multiple inheritance:
    def __init__(self):
        print(" This is class C")
        A.__init__(self)
        B.__init__(self)

obj=C()
```

```
This is class C
This is class A
This is class B
```

encapsulation

In [57]:
```
# encapsulation is wrapping of data under single unit.<public private and protected>
```

In [58]:
```
# public method:can access anywhrer from outside of classa
class employe:
    def __init__(self,name,salary,age):
        self.name=name
        self.salary=salary
        self.age=age

    def show(self):
        print('name:',self.name,'salary:',self.salary,"age:",self.age)
emp=employe('hsn','200','30')
emp.show()
print("salary:",emp.salary)
```

```
name: hsn salary: 200 age: 30
salary: 200
```

In [59]:
```
## Accessible modifier in python(encapsulation:)
#--> Public Member: Accessible anywhere from otside oclass.
#--> Private Member: Accessible within the class (use double underscore)
#--> Protected Member: Accessible within the class and its sub-classes
class employe:
    def __init__(self,name,salary,age):
        self.__name="ali" # private attribute
        self.salary=salary
        self.age=age

    def show(self):
        print('name:',self.__name,'salary:',self.salary,"age:",self.age)
emp=employe('hsn','200','30')
print("name",emp.__name)# cannot access directly from outside of class:
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_13068/2499310826.py in <module>
     12         print('name:',self.__name,'salary:',self.salary,"age:",self.age)
     13 emp=employe('hsn','200','30')
---> 14 print("name",emp.__name)# cannot access directly from outside of class:

AttributeError: 'employe' object has no attribute '__name'
```

In [60]:
```
# name mangling are used to access attribute in private class:
# we can also create private method:
class employe:
    def __init__(self,name,salary,age):
        self.__person()
        self.__name=name # private attribute
        self.salary=salary
        self.age=age

    def show(self):
        print('name:',self.__name,'salary:',self.salary,"age:",self.age)
    def __person(self): # private function
        print("This is person employee")
emp=employe('hsn','200','30')
emp.show()
print("name:",emp._employe__name)# name-mangling used here to access
(private-class) attrbute of class:
emp._employe__person() # name mangling
```

```
This is person employee
name: hsn salary: 200 age: 30
name: hsn
This is person employee
```

In [61]:
```
# protected: access within class and sub_class
class employe:
    def __init__(self):
         # protected attribute
        self.salary="2000"
        self._age="16"

class worker(employe):
    def __init__(self,name):
        self._name=name
        employe.__init__(self)

    def show(self):


print('name:',self._name,'salary:',self.salary,"age:",self._age)
```

```
w=worker('ali')
w.show()
#print("name",w._worker__name)# cannot access directly from outside of
class:
print(w._age)
```

```
name: ali salary: 2000 age: 16
16
```

In [62]:

```
# getter and setter
class employ:
    def __init__(self,name,roll):
        self.name=name
        self.roll=roll
        print(self.name,roll)
    def set_age(self,age):
        self.age=age
    def get_age(self):
        return self.age

emp=employ('ali','122')
emp.set_age("35")
print(emp.get_age())
emp.roll # print single attribute from class.
```

```
ali 122
35
```

Out[62]:

```
'122'
```

In [63]:

```
class car:
    def __init__(self):
        self.software('Android','2021')
        # method declaration in constructor class:(driver & software)
        self.drive()
    def drive(self):
        print("this is driver")
    def software(self,name,id):
        self.name=name
        self.id=id
        print("software updates:",self.name,id)
class truc(car):
```

```
    def __init__(self):
        car.__init__(self)# parent class initilization
        print("this truc")
t=truc()
```

```
software updates: Android 2021
this is driver
this truc
```

polymorphism

same function name can be used for different types. This makes programming more intuitive and easier.

In [64]:

```
# polymorphism in inheritance
# differnet operation in single form:
class car:
    def engine(self):
        print("this car have ivvt engine")
class van(car):

    def engine(self):
        super().engine() # super function used to access car method
        print("this van have ivvt engine")
obj=van()
obj.engine() # this is polymorphism
```

```
this car have ivvt engine
this van have ivvt engine
```

In [65]:

```
 #concept of polymorphism in classes and object:
#function have same name called method overriding
class pakistan:
    def area(self):
        print("pakistan have 200000 area")
    def capital(self):
        print("pk-->islamabad")
class india:
    def area(self):
```

```
        print("india have 900000 area")
    def capital(self):
        print("in-->new dehli")
pk=pakistan()
inr=india()
for detail in (pk,inr):
    detail.area()
    detail.capital()
```

```
pakistan have 200000 area
pk-->islamabad
india have 900000 area
in-->new dehli
```

method overloading and overriding

In [66]:

```
#It simply refers to the use of numerous methods within a class with same
name
#but accepting different number of arguments
class clalculation:
    def sum(self,a=None,b=None,c=None): # method overloading
        s=0
        if a !=None and b !=None and c !=None:
            s=a+b+c
        elif a !=None and b !=None:
            s=a+b
        else:
            s=a
        return s

    def multiply(self,a,b):
        m=a*b
        return m
    def multiply(self,a,b,c):
        m=a*b*c
        return m
c=clalculation()
print(c.sum(10))
print(c.sum(10,40))
print(c.multiply(2,8,9))
```

```
10
50
144
```

In [67]:

```
# method overriding in inheritance
class methodOverride1:
    def display(self):
        print("method invoked from base class")

class methodOverride2(methodOverride1):
    def display(self):
        print("method invoked from derived class")

ob=methodOverride2()
ob.display()
```

```
method invoked from derived class
```

Bike rental system

In [69]:

```
 total=100
while True:
#     print('''
#     1-for total bike
#     2-for rent a bike(per 100)
#     3-exit
#     ''')
    opt=int(input('''
    1-for total bike
    2-for rent a bike(per 100)
    3-exit
    '''))
    if opt==1:
        print("total bike available",total)
    if opt==2:
        bill=0
        rent=int(input(' the quantity of bikes:'))
        if rent>100:
            print("Bike not available more then 100")
            continue
        total=total-rent
```

```
        print("resrved bikes is :",total)
        bill=rent*100
        print("your total bill is:",bill)
    if opt==3:
        break
```

```
   1-for total bike
   2-for rent a bike(per 100)
   3-exit
   1
total bike available 100

   1-for total bike
   2-for rent a bike(per 100)
   3-exit
   2
 the quantity of bikes:50
resrved bikes is : 50
your total bill is: 5000

   1-for total bike
   2-for rent a bike(per 100)
   3-exit
   3
```