Day Objectives(7/9/19)

Functional Programming (high efficiency(to reduce lines of code))

List Comprehension

Iterators and Generators

Map

Filters

Lambda

In [2]:
```
##List comprehension
##list compression is very easy way to create list
##compared with for loop (more efficient than list with for loop)
# program using list comprehension
n=10
li=[]
for i in range(1,n+1):
    li.append(i)
print(li)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

In [3]:
```
##using list comprehension
# list comprehension ->[]
# left side ones are generally what we print
li=[i for i in range(1,11)]  #functional programming to reduce num of lines
li
```

Out[3]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

In [8]:
```
#sqroot
def sqroot():
    l=[(i**(1/2))for i in li]
    for i in l:
        print("sqrt:",i)
sqroot()
```

```
sqrt: 1.0
sqrt: 1.4142135623730951
sqrt: 1.7320508075688772
sqrt: 2.0
sqrt: 2.23606797749979
sqrt: 2.449489742783178
sqrt: 2.6457513110645907
sqrt: 2.8284271247461903
sqrt: 3.0
sqrt: 3.1622776601683795
```

In [7]:
```
#cuberoot
def cuberoot():
    l=[(i**(1/3))for i in li]
    for i in l:
        print("cubert:",i)
cuberoot()
```

```
cubert: 1.0
cubert: 1.2599210498948732
cubert: 1.4422495703074083
cubert: 1.5874010519681994
cubert: 1.7099759466766968
cubert: 1.8171205928321397
cubert: 1.912931182772389
cubert: 2.0
cubert: 2.080083823051904
cubert: 2.154434690031884
```

In [2]:
```
##Leap year(list comprehension)
lb=1970
ub=2020
leapyear=[i for i in range(lb,ub+1) if(i%400==0) or (i%100!=0 and i%4==0)]
leapyear
```

Out[2]:
```
[1972, 1976, 1980, 1984, 1988, 1992, 1996, 2000, 2004, 2008, 2012, 2016, 2020]
```

In [3]:
```
lb=int(input())
ub=int(input())
leapyear=[i for i in range(lb,ub+1) if(i%400==0) or (i%100!=0 and i%4==0)]
leapyear
```

```
1940
2010
```

Out[3]:
```
[1940,
 1944,
 1948,
 1952,
 1956,
 1960,
 1964,
 1968,
 1972,
 1976,
 1980,
 1984,
 1988,
 1992,
 1996,
 2000,
 2004,
 2008]
```

In [21]:
```
##cummulative sum
def cumsum(n):
    sum=0
    li=[]
    for i in range(1,n+1):
        sum=sum+i
        print(sum)
        li.append(sum)
    print(li)
cumsum(4)
```

```
1
3
6
10
[1, 3, 6, 10]
```

In [23]:
```
##cummulative mult
def cummult(n):
    mult=1
    li=[]
    for i in range(1,n+1):
        mult=mult*i
        print(mult)
        li.append(mult)
    print(li)
cummult(4)
```

```
1
2
6
24
[1, 2, 6, 24]
```

In [34]:
```
##cummulative sum (list comprehension)
n=7
cumsum1=[sum(range(1,i+1)) for i in range(1,n+1)]
cumsum1
```

Out[34]:
```
[1, 3, 6, 10, 15, 21, 28]
```

In [41]:
```
import numpy
##cummulative mult (list comprehension)
n=7
cummul=[numpy.prod(range(1,i+1)) for i in range(1,n+1)]
cummul
```

Out[41]:
```
[1, 2, 6, 24, 120, 720, 5040]
```

Iterators (It is also functional programming,used to reduce lines of code)

iterator is an object that contains number of elements

2-methods

iter()

next()

all ds are all iterable objects

#for data structures we need not use loop,my default it will take(all data structures are iterable objects)

#ds have 2 parts -> 1.iter 2.next

In [42]:
```
dir(iter)
```

Out[42]:
```
['__call__',
 '__class__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__name__',
 '__ne__',
 '__new__',
 '__qualname__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__self__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__text_signature__']
```

In [61]:
```
##iter,next operationsit="python"
it='python'
it=iter("python")
print(it)##python memory will be displayed
print(next(it))
print(next(it))
print(it)
print(it)
print(next(it))
print(it)
print(it)
print(next(it))
print(next(it))
print(it)
print(next(it))
```

```
<str_iterator object at 0x000002C0D7486A58>
p
y
<str_iterator object at 0x000002C0D7486A58>
<str_iterator object at 0x000002C0D7486A58>
t
<str_iterator object at 0x000002C0D7486A58>
<str_iterator object at 0x000002C0D7486A58>
h
o
<str_iterator object at 0x000002C0D7486A58>
n
```

In [62]:
```
it='python'
it=iter("python")
print(it)##python memory will be displayed
print(next(it))
print(it)
print(it)
print(next(it))
print(it)
print(it)
print(next(it))
print(next(it))
print(it)
print(next(it))
```

```
<str_iterator object at 0x000002C0D7486748>
p
<str_iterator object at 0x000002C0D7486748>
<str_iterator object at 0x000002C0D7486748>
y
<str_iterator object at 0x000002C0D7486748>
<str_iterator object at 0x000002C0D7486748>
t
h
<str_iterator object at 0x000002C0D7486748>
o
```

In [59]:
```
#same above program by using for loop
it='python'
for i in it:
    print(i)
```

```
p
y
t
h
o
n
```

In [82]:
```
##method 1 w/o using for loop(b)
a="python"
b=iter(a)
print(next(b))
print(next(b))
print(next(b))
print(next(b))
print(next(b))
print(next(b))
```

```
p
y
t
h
o
n
```

In [74]:
```
##method-2 without using for loop (by using list)
a=["python","pythonprogram"]
b=iter(a)
print(next(b))
print(next(b))
```

```
python
pythonprogram
```

In [85]:
```
##method1 sample error
a=["python","pythonprogram"]
b=iter(a)
print(next(b))
print(next(b))
print(next(b))
print(next(b))
print(next(b))
print(next(b))
```

```
python
pythonprogram

---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-85-f50272862161> in <module>()
      4 print(next(b))
      5 print(next(b))
----> 6 print(next(b))
      7 print(next(b))
      8 print(next(b))

StopIteration:
```

In [86]:
```
##method2 sample error
a=["python","pythonprogram"]
b=iter(a)
print(next(b))
print(next(b))
print(next(b))
print(next(b))
print(next(b))
print(next(b))
```

```
python
pythonprogram

---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-86-f82475cdb377> in <module>()
      4 print(next(b))
      5 print(next(b))
----> 6 print(next(b))
      7 print(next(b))
      8 print(next(b))

StopIteration:
```

In [90]:
```
#generator is fun we can create iteratives
#we use yield keyword to return values just like return in function #generator is
def generatorfun(): #yield in generator is like return in function #iter is repla
    yield 1
    yield 2
    yield 4
print(generatorfun())
for value in generatorfun():
    print(value)
```

```
<generator object generatorfun at 0x000002C0D8C16FC0>
1
2
4
```

In [91]:
```
#sq of num in given range (using generators)
#next(a) for printing line by line instead of printing in same line
def gennum():
    n=3
    while True:
        n**=3
        yield n
a=gennum()
for i in range(5):
    print(next(a))
```

```
27
19683
7625597484987
443426488243037769948249630619149892803
871896424859609582029110705858607716969640724047317500855252194379909670937234 3
994347554990683168311679105522566562 7
```

Maps in python

In [104]:
```
### It produces a list of results that it apply given function of each item of a 
### syntax: map(function,iterable)
def add(n):
    return n+n
#1+1=2,2+2=4,3+3=6,4+4=8
result=list(map(add,range(1,5))) ##to get output in list format we need to use li
print(result)
```

```
[2, 4, 6, 8]
```

In [106]:
```
###same above one w/o using list ,map
def add(n):
    result=for i in range(1,n) ##to get output in list format we need to use list
      return n+n
add(7)
```

```
  File "<ipython-input-106-f93b3c703c89>", line 3
    result=for i in range(1,n) ##to get output in list format we need to use li
st map
             ^
SyntaxError: invalid syntax
```

In [112]:
```
#mul using map
#eg 1
def prod(k):
    return k*2
#1+1=2,2+2=4,3+3=6,4+4=8
result=list(map(prod,range(1,5))) ##to get output in list format we need to use l
print(result)
```

```
[2, 4, 6, 8]
```

In [117]:
```
#eg2 sq
def sqnum(n):
    return n*n
result=set(map(sqnum,range(1,5)))
print(result)
```

```
{16, 1, 4, 9}
```

In [118]:
```
#eg3 sq
def sqnum(n):
    return n**2
result=set(map(sqnum,range(1,5)))
print(result)
```

```
{16, 1, 4, 9}
```

In [119]:
```
#eg4
#sqrt
def sqroot(n):
    return n**0.5
result=set(map(sqroot,range(1,5)))
print(result)
```

```
{1.0, 2.0, 1.7320508075688772, 1.4142135623730951}
```

Filters in python

In [133]:
```
#method not a function(it checks each function,boolean also checks but not for ev
#The filter() function in Python takes in a function and a list as arguments.
#This offers an elegant way to filter out all the elements of a sequence "sequenc
#We can use Lambda function inside the filter() built-in function to find all the
#In Python, anonymous function means that a function is without a name.
li=[1,2,'a','b','c',3]
def isdigit(c):
    c=str(c)
    if c.isdigit():
        return True
    return False
isdigit('a')
list(filter(isdigit,li))
```

Out[133]:
```
[1, 2, 3]
```

In [131]:
```
#sample error
##above ex if we donot give str, we will get error(if we donot convert int to str
#in list we have combination of str and int(so we are converting all to string fo
##AFTER USING FILTER WE SHOULD NOT USE FUNCTION
## syntax: f:x->{T<F}
li=[1,2,'a','b','c',3]
def isdigit(c):
    if c.isdigit():
        return True
    return False
isdigit('a')
list(filter(isdigit,li))
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-131-8735dbfc168d> in <module>()
      8     return False
      9 isdigit('a')
---> 10 list(filter(isdigit,li))

<ipython-input-131-8735dbfc168d> in isdigit(c)
      4 li=[1,2,'a','b','c',3]
      5 def isdigit(c):
----> 6     if c.isdigit():
      7         return True
      8     return False

AttributeError: 'int' object has no attribute 'isdigit'
```

In [134]:
```
#sample error
def isdigit(c):
    if c.isdigit():
        return True
    return False
isdigit()#if we donot give any parameters then we will get error
list(filter(isdigit,li))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-134-e8af7ab7accd> in <module>()
      4         return True
      5     return False
----> 6 isdigit()#if we donot give any parameters then we will get error
      7 list(filter(isdigit,li))

TypeError: isdigit() missing 1 required positional argument: 'c'
```

In [ ]:

NUMPY & PANDAS

- Numpy

(Base N-Dimensional array package)

- numpy is a python library and a fundamental packages for scientific computing in python
- It is used to create multidimentional arrays for fast operations
- Use the following import conversation to access numpy library (import numpy as np)
- Have a fixed size of arrays used to store a collection of elements,unlike,python lists
- They three types of arrays called 1D 2D 3D

In [156]:
```
#creating 1D array using numpy#numpy is used to create multidimensional arrays
import numpy as np#instead of np we can give anything # as refers to alias
array_1d=np.array([2,4,5,6,7,9])
print(array_1d)
type(array_3d)
```

```
[2 4 5 6 7 9]
```

Out[156]:
```
numpy.ndarray
```

In [157]:
```
##creating 2D array
import numpy as np
array_2d=np.array([(2,4,5),(6,7,9)])
print(array_2d)
type(array_3d)
```

```
[[2 4 5]
 [6 7 9]]
```

Out[157]: numpy.ndarray

In [158]:
```
##creating 3D array
import numpy as np
array_3d=np.array([(2,4,5),(6,7,9),(2,2,2)])
print(array_3d)
type(array_3d)
```

```
[[2 4 5]
 [6 7 9]
 [2 2 2]]
```

Out[158]: numpy.ndarray

In [160]:
```
##creating 3D array(if we donot give same num of elements in all,then it gives re
import numpy as np
array_3d=np.array([(2,4,5),(6,7,9),(1,1,)])
print(array_3d)
type(array_3d)
```

```
[(2, 4, 5) (6, 7, 9) (1, 1)]
```

Out[160]: numpy.ndarray

In [153]:
```
#Program to print the given list in to an array
import numpy as np
li=[1,2,3,4,'b','z']
a=np.array(li)
print(a)
```

```
['1' '2' '3' '4' 'b' 'z']
```

In [162]:
```
# example
# To print 1D array between 1 to 15
import numpy as np
a=np.arange(15)
print(a)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]
```

In [165]:
```
import numpy as np
a=np.arange(15)
a
```

Out[165]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

In [175]:
```
import numpy as np
a=np.arange(-1)
a
```

Out[175]:
```
array([], dtype=int32)
```

In [178]:
```
import numpy as np #arange->it is array range
a=np.arange(10000)#IF WE GIVE MORE THAN 1000 IT WILL SKIP MIDDLE VALUES
a
```

Out[178]:
```
array([   0,    1,    2, ..., 9997, 9998, 9999])
```

In [184]:
```
##ex1
a=np.arange(10000).reshape(5000,2)
print(a)
```

```
[[   0    1]
 [   2    3]
 [   4    5]
 ...
 [9994 9995]
 [9996 9997]
 [9998 9999]]
```

In [187]:
```
##ex1
a=np.arange(0).reshape(0,8)
print(a)
```

```
[]
```

In [190]:
```
##ex1
a=np.arange(100).reshape(50,2)
a
```

Out[190]:
```
array([[ 0,  1],
       [ 2,  3],
       [ 4,  5],
       [ 6,  7],
       [ 8,  9],
       [10, 11],
       [12, 13],
       [14, 15],
       [16, 17],
       [18, 19],
       [20, 21],
       [22, 23],
       [24, 25],
       [26, 27],
       [28, 29],
       [30, 31],
       [32, 33],
       [34, 35],
       [36, 37],
       [38, 39],
       [40, 41],
       [42, 43],
       [44, 45],
       [46, 47],
       [48, 49],
       [50, 51],
       [52, 53],
       [54, 55],
       [56, 57],
       [58, 59],
       [60, 61],
       [62, 63],
       [64, 65],
       [66, 67],
       [68, 69],
       [70, 71],
       [72, 73],
       [74, 75],
       [76, 77],
       [78, 79],
       [80, 81],
       [82, 83],
       [84, 85],
       [86, 87],
       [88, 89],
       [90, 91],
       [92, 93],
       [94, 95],
       [96, 97],
       [98, 99]])
```

In [192]:
```
p=range(1000)
p
```

Out[192]:
```
range(0, 1000)
```

In [197]:
```
n=range(1000)
e.[i**2 for i in range(1000)]
print(e)
```

```
  File "<ipython-input-197-da408b119077>", line 2
    e.[i**2 for i in range(1000)]
      ^
SyntaxError: invalid syntax
```

In [216]:
```
np.arange(3,10,2)
```

Out[216]:
```
array([3, 5, 7, 9])
```

In [204]:
```
np.zeros((3,2))
```

Out[204]:
```
array([[0., 0.],
       [0., 0.],
       [0., 0.]])
```

In [208]:
```
np.ones((3,2))
```

Out[208]:
```
array([[1., 1.],
       [1., 1.],
       [1., 1.]])
```

In [215]:
```
np.eye(3)
```

Out[215]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

In [217]:
```
#it will be valid for ones,zeros
np.twos((3,2))
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-217-4ca9efd436d9> in <module>()
      1 #it will be valid for ones,zeros
----> 2 np.twos((3,2))

AttributeError: module 'numpy' has no attribute 'twos'
```

In [222]:
```
np.full((3,3),2.5)#it takes int,float values
```

Out[222]:
```
array([[2.5, 2.5, 2.5],
       [2.5, 2.5, 2.5],
       [2.5, 2.5, 2.5]])
```

In [224]:
```
##all float values convert to integers
np.full((3,3),4.4,dtype=np.int)
```

Out[224]:
```
array([[4, 4, 4],
       [4, 4, 4],
       [4, 4, 4]])
```

In [228]:
```
#diagonal
np.diag([1,9,9,9,28])
```

Out[228]:
```
array([[ 1,  0,  0,  0,  0],
       [ 0,  9,  0,  0,  0],
       [ 0,  0,  9,  0,  0],
       [ 0,  0,  0,  9,  0],
       [ 0,  0,  0,  0, 28]])
```

In [229]:
```
f=np.array([2,3,4,5])
f
```

Out[229]:
```
array([2, 3, 4, 5])
```

In [230]:
```
#horizontally(rows) 3lines filled with f[list],vertically 1 column
np.full((a,b),c)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-230-aa45d354a680> in <module>()
      1 #horizontally(rows) 3lines filled with f[list],vertically 1 column
----> 2 np.full((a,b),c)

NameError: name 'c' is not defined
```

In [309]:
```
f=range(1000)
```

In [311]:
```
f
```

Out[311]:
```
range(0, 1000)
```

In [313]:
```
#dynamically
a=int(input("enter row:"))
b=int(input("enter col:"))
c=int(input("enter num:"))
np.full((a,b),c)
```

```
enter row:4
enter col:3
enter num:5
```

Out[313]:
```
array([[5, 5, 5],
       [5, 5, 5],
       [5, 5, 5],
       [5, 5, 5]])
```

In [231]:
```
f=[1,2,3]
f1=np.array(f)
f1
```

Out[231]:
```
array([1, 2, 3])
```

In [233]:
```
#horizontally(rows) 3lines filled with f[list],vertically 1 column
np.tile(f1,(5,1))
```

Out[233]:
```
array([[1, 2, 3],
       [1, 2, 3],
       [1, 2, 3],
       [1, 2, 3],
       [1, 2, 3]])
```

In [234]:
```
50*np.random.random() +2  #2 to 50
```

Out[234]:
```
28.983190044773607
```

In [235]:
```
np.random.random()  #returns random
```

Out[235]:
```
0.1121877889998032
```

In [236]:
```
np.random.random((3,3))
```

Out[236]:
```
array([[0.62275078, 0.63023467, 0.27330768],
       [0.43387558, 0.49619735, 0.52846199],
       [0.35936055, 0.85591917, 0.72796182]])
```

In [241]:
```
p=np.linspace(1,50,23) #1 to 50 num divided into 23 parts
p
```

Out[241]:
```
array([ 1.        ,  3.22727273,  5.45454545,  7.68181818,  9.90909091,
       12.13636364, 14.36363636, 16.59090909, 18.81818182, 21.04545455,
       23.27272727, 25.5       , 27.72727273, 29.95454545, 32.18181818,
       34.40909091, 36.63636364, 38.86363636, 41.09090909, 43.31818182,
       45.54545455, 47.77272727, 50.        ])
```

In [242]:
```
p.itemsize
```

Out[242]:
```
8
```

In [249]:
```
z=np.arange(18).reshape(1,6,3)
z
```

Out[249]:
```
array([[[ 0,  1,  2],
        [ 3,  4,  5],
        [ 6,  7,  8],
        [ 9, 10, 11],
        [12, 13, 14],
        [15, 16, 17]]])
```

In [254]:
```
np.shares_memory(a,p)
```

Out[254]:
```
False
```

In [250]:
```
z=np.arange(18).reshape(2,3,3)
z
```

Out[250]:
```
array([[[ 0,  1,  2],
        [ 3,  4,  5],
        [ 6,  7,  8]],

       [[ 9, 10, 11],
        [12, 13, 14],
        [15, 16, 17]]])
```

In [256]:
```
z.shape
```

Out[256]:
```
(2, 3, 3)
```

In [ ]:

Subset of array

In [ ]:

PANDAS

- pandas are basically use for data analysis.It is build on top of numpy only
- we will mostly be using this pandas package a lot
- In fact if you are going to use it heavily for any data manipulation,anysort of data analysis and any sort of datavisualisation ,any sort of machine learing model building ##### DataVisualisation

In [278]:
```
import pandas as pd
```

In [279]:
```
a=pd.Series([1,2,3,4,5])  #pandas also homogeneousarray
```

In [280]:
```
a
```

Out[280]:
```
0    1
1    2
2    3
3    4
4    5
dtype: int64
```

In [281]:
```
type(a)
```

Out[281]:
```
pandas.core.series.Series
```

In [283]:
```
a[2]
```

Out[283]:
```
3
```

In [287]:
```
a=pd.Series(['a','b','c'])  #pandas also homogeneousarray
```

In [292]:
```
a
```

Out[292]:
```
0    a
1    b
2    c
dtype: object
```

In [293]:
```
#data datatype: to convert x date to days
a=pd.date_range(start="8-1-19",end='20-8-19')
```

In [294]:
```
a
```

Out[294]:
```
DatetimeIndex(['2019-08-01', '2019-08-02', '2019-08-03', '2019-08-04',
               '2019-08-05', '2019-08-06', '2019-08-07', '2019-08-08',
               '2019-08-09', '2019-08-10', '2019-08-11', '2019-08-12',
               '2019-08-13', '2019-08-14', '2019-08-15', '2019-08-16',
               '2019-08-17', '2019-08-18', '2019-08-19', '2019-08-20'],
              dtype='datetime64[ns]', freq='D')
```

Pandas DataFrame

- Dataframe is just like a table.which is going to contain rows and column
- Row can have a index
- Column can have a meaning full names
- (pandas as dataframe)

In [295]:
```
import numpy as np
```

In [296]:
```
temp=np.random.randint(low=20,high=100,size=[20])
name=np.random.choice(['divija','divya','divi','geethu'],20)
av=np.random.choice([10,23,64,24,41],20)
```

In [298]:
```
a=list(zip(temp,name,av))
```

In [306]:
```
df=pd.DataFrame(data=a,columns=['temp','name','av'])
```

In [307]:
```
df
```

Out[307]:

	temp	name	av
0	41	divya	41
1	55	geethu	23
2	37	divya	41
3	58	divi	64
4	43	divi	10
5	79	divya	41
6	89	divi	24
7	52	divija	24
8	67	geethu	41
9	54	geethu	23
10	46	geethu	24
11	87	divya	23
12	36	divija	24
13	30	divya	24
14	95	divi	23
15	93	divi	64
16	32	divija	64
17	75	divi	64
18	50	divi	64
19	98	divya	24

In [320]:
```
#to generate random value
import numpy as np
temp=np.random.randint(low=20,high=100,size=[20])
temp
```

Out[320]:
```
array([33, 47, 23, 35, 91, 84, 71, 41, 41, 77, 52, 55, 46, 59, 83, 89, 24,
       40, 41, 48])
```

In [321]:
```
name=np.random.choice(['divija','divya','divi','geethu'],20)
name
```

Out[321]:
```
array(['divija', 'divija', 'divi', 'divya', 'divya', 'divija', 'divi',
       'divi', 'geethu', 'divi', 'divi', 'geethu', 'geethu', 'geethu',
       'geethu', 'divi', 'geethu', 'geethu', 'divya', 'divya'],
      dtype='<U6')
```

In [322]:
```
name=np.random.choice(['divija','divya','divi','geethu'])
name
```

Out[322]:
```
'geethu'
```

In [324]:
```
type(df)
```

Out[324]:
```
pandas.core.frame.DataFrame
```

In [327]:
```
#first 5 values
df.head()
```

Out[327]:

	temp	name	av
0	41	divya	41
1	55	geethu	23
2	37	divya	41
3	58	divi	64
4	43	divi	10

In [328]:
```
#last 5 values
df.tail()
```

Out[328]:

	temp	name	av
15	93	divi	64
16	32	divija	64
17	75	divi	64
18	50	divi	64
19	98	divya	24

In [335]:
```
#size of dataframe
df.shape
```

Out[335]:
```
(20, 3)
```

In [336]:
```
df.columns
```

Out[336]:
```
Index(['temp', 'name', 'av'], dtype='object')
```

In [337]:
```
df.name
```

Out[337]:
```
0      divya
1     geethu
2      divya
3       divi
4       divi
5      divya
6       divi
7     divija
8     geethu
9     geethu
10    geethu
11     divya
12    divija
13     divya
14      divi
15      divi
16    divija
17      divi
18      divi
19     divya
Name: name, dtype: object
```

In [340]:
```
df.temp
```

Out[340]:
```
0     41
1     55
2     37
3     58
4     43
5     79
6     89
7     52
8     67
9     54
10    46
11    87
12    36
13    30
14    95
15    93
16    32
17    75
18    50
19    98
Name: temp, dtype: int64
```

In [341]:
```
df.av
```

Out[341]:
```
0     41
1     23
2     41
3     64
4     10
5     41
6     24
7     24
8     41
9     23
10    24
11    23
12    24
13    24
14    23
15    64
16    64
17    64
18    64
19    24
Name: av, dtype: int64
```

In [344]:
```
df['temp'].describe()
```

Out[344]:
```
count    20.000000
mean     60.850000
std      22.725536
min      30.000000
25%      42.500000
50%      54.500000
75%      81.000000
max      98.000000
Name: temp, dtype: float64
```

In [345]:
```
df['name'].describe()
```

Out[345]:
```
count       20
unique       4
top       divi
freq         7
Name: name, dtype: object
```

In [346]:
```
df['av'].describe()
```

Out[346]:
```
count    20.000000
mean     36.500000
std      18.109317
min      10.000000
25%      23.750000
50%      24.000000
75%      46.750000
max      64.000000
Name: av, dtype: float64
```

In [347]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 3 columns):
temp    20 non-null int64
name    20 non-null object
av      20 non-null int64
dtypes: int64(2), object(1)
memory usage: 560.0+ bytes
```

In [349]:
```
df.set_index('temp')
```

Out[349]:

temp	name	av
41	divya	41
55	geethu	23
37	divya	41
58	divi	64
43	divi	10
79	divya	41
89	divi	24
52	divija	24
67	geethu	41
54	geethu	23
46	geethu	24
87	divya	23
36	divija	24
30	divya	24
95	divi	23
93	divi	64
32	divija	64
75	divi	64
50	divi	64
98	divya	24

In [350]:
```
df.set_index('name')
```

Out[350]:

name	temp	av
divya	41	41
geethu	55	23
divya	37	41
divi	58	64
divi	43	10
divya	79	41
divi	89	24
divija	52	24
geethu	67	41
geethu	54	23
geethu	46	24
divya	87	23
divija	36	24
divya	30	24
divi	95	23
divi	93	64
divija	32	64
divi	75	64
divi	50	64
divya	98	24

In [351]:
```
df.set_index('av')
```

Out[351]:

av	temp	name
41	41	divya
23	55	geethu
41	37	divya
64	58	divi
10	43	divi
41	79	divya
24	89	divi
24	52	divija
41	67	geethu
23	54	geethu
24	46	geethu
23	87	divya
24	36	divija
24	30	divya
23	95	divi
64	93	divi
64	32	divija
64	75	divi
64	50	divi
24	98	divya

In [359]:
```
#To concatinate row wise
name=np.random.choice(['geethuuu','divijaaa','loyola'],9)
name1=np.random.choice(['divija','divya','divi','geethu'],20)
sno=np.random.randint(0,9,5)
print(name)
print(name1)
print(sno)
```

```
['geethuuu' 'loyola' 'divijaaa' 'divijaaa' 'loyola' 'loyola' 'loyola'
 'loyola' 'loyola']
['divi' 'divya' 'divija' 'divya' 'divi' 'divya' 'divi' 'divya' 'divi'
 'geethu' 'divija' 'divya' 'divija' 'divi' 'divya' 'geethu' 'divi'
 'divija' 'divya' 'divya']
[3 0 8 8 5]
```

In [381]:
```
import pandas as pd
name=np.random.choice(['geethu','chaya','revanth','ivic','shreela','bhavvuuu'],10
R=np.random.choice(['friend','bff','sis','bro'],10)
sn0=np.random.randint(0,10,10)
bond=np.random.randint(90,100,10)
b=list(zip(sn0,name,R,bond))
df1=pd.DataFrame(data=b,columns=['sno','name','R','bond'])
df1
```

Out[381]:

	sno	name	R	bond
0	5	shreela	friend	90
1	6	revanth	friend	95
2	3	bhavvuuu	friend	97
3	8	revanth	sis	91
4	3	shreela	bff	90
5	4	chaya	bff	96
6	6	shreela	bro	91
7	5	revanth	friend	94
8	3	bhavvuuu	friend	97
9	0	revanth	sis	90

In [380]:
```
pd.concat([d1,d2],axis=0)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-380-1aaf0d7b08a7> in <module>()
----> 1 pd.concat([d1,d2],axis=0)

NameError: name 'd1' is not defined
```

Data visualisation

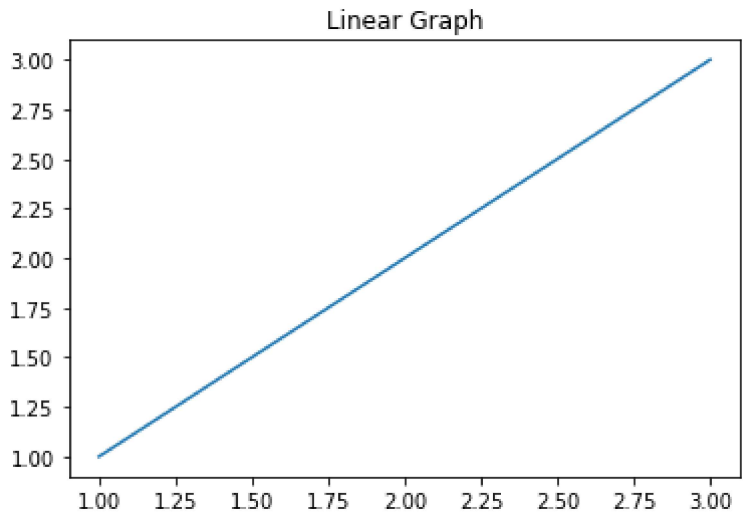
- Matplotlib is a plotting library for the python programming language and its numerical mathematical extensions numpy
- Data in a graphical format or in a graphical representation
- Different types of graph representation
  - Line graph(linear graph)
  - Bar graph
  - Histogram

Line graph or linear graph

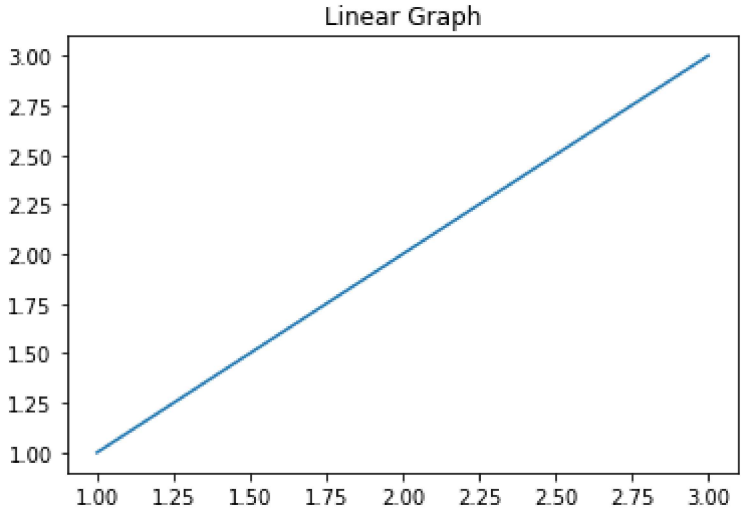
Representation of linear graph

- line graph is a type of graph which displays info's as a series of data points called'markers'connected by st. line segments
- line graphs are usually used to find relationship between 2data sets on different axis;for instance X,Y

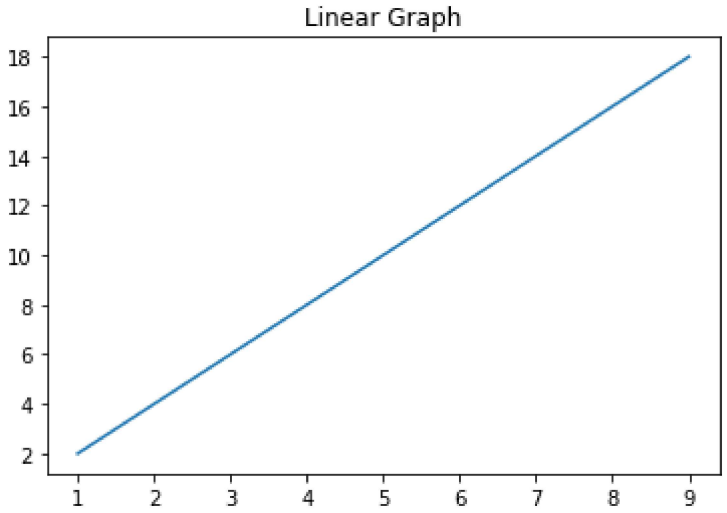
In [365]:
```
from matplotlib import pyplot as plt
plt.title("Linear Graph")
plt.plot([1,2,3],[1,2,3])
plt.show()
```



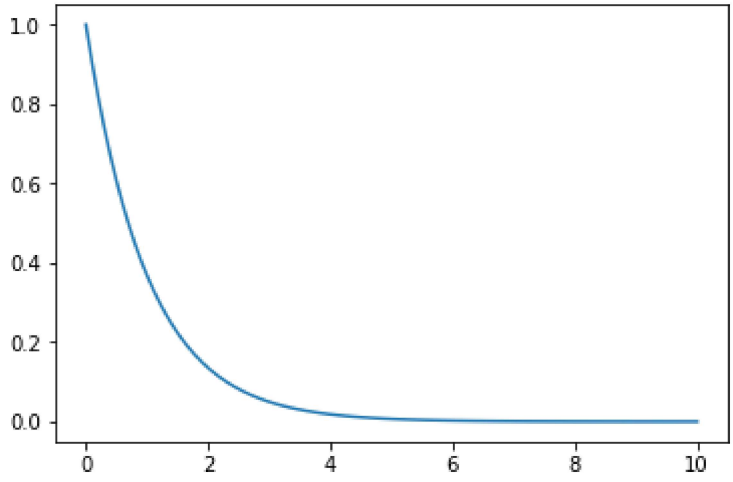
In [388]:
```
from matplotlib import pyplot as plt
plt.title("Linear Graph")
plt.plot([1,2,3],[1,2,3])
plt.show()
```



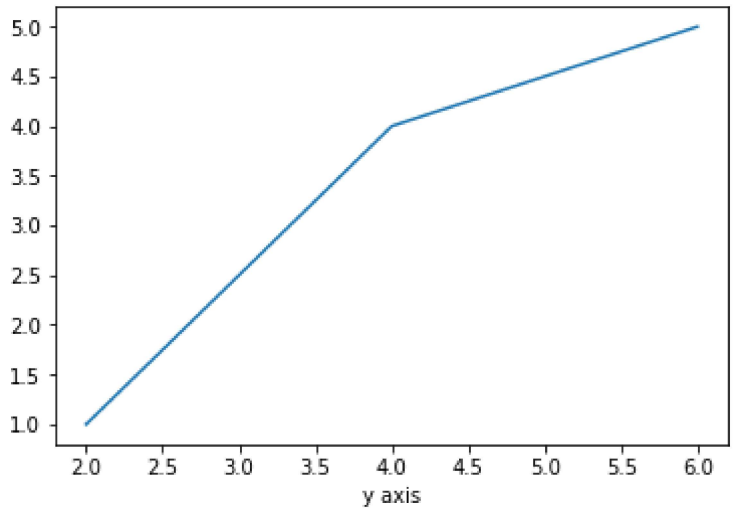
In [386]:
```
from matplotlib import pyplot as plt
plt.title("Linear Graph")
plt.plot([1,7,9],[2,14,18])
plt.show()
```



In [374]:
```
import matplotlib.pyplot as plt
import numpy as np
a=np.linspace(0,10,100)
b=np.exp(-a)
plt.plot(a,b)
plt.show()
```

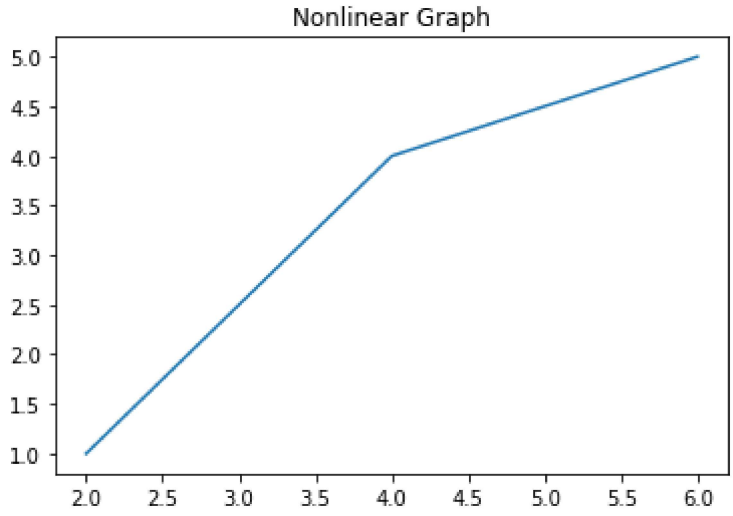


In [395]:
```
from matplotlib import pyplot as plt
x=[2,4,6]
y=[1,4,5]
plt.plot(x,y)
plt.xlabel("x axis")
plt.xlabel("y axis")
plt.show()
```



Non-linear Graph

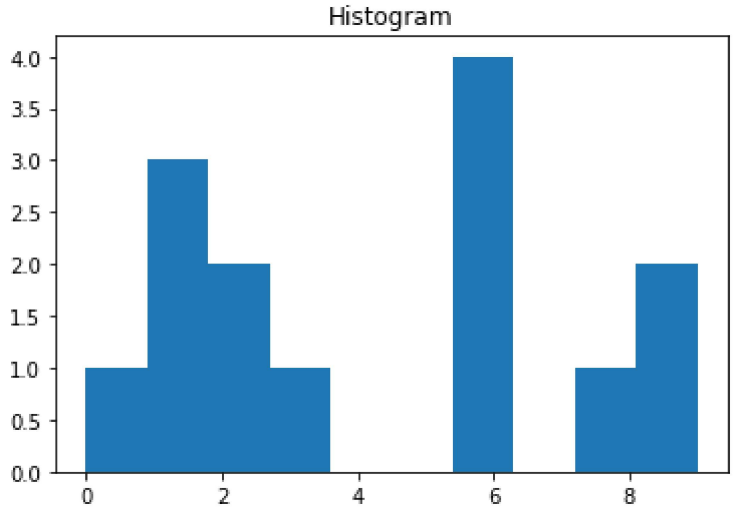
In [396]:
```
from matplotlib import pyplot as plt
x=[2,4,6]
y=[1,4,5]
plt.title("Nonlinear Graph")
plt.plot(x,y)
plt.show()
```



Histogram

- A histogram shows freq on the vertical axis and horizontal axis is another direction

In [397]:
```
from matplotlib import pyplot as plt
y=[1,2,2,6,6,6,8,9,9,0,1,1,3,6]
plt.title("Histogram")
plt.hist(y)#fun to plot histogram
plt.show()#fun to show plot
```



Lambda in python

-(continuation after filters)

(no need to give fun name,it works for small fun's)(lambda is nopt iterable ,so we use iter,map,...)(it can take 0 or more arg,but 1 exp,that is why it doesnot work for large programs)

- python lambda is just a function w/o name or anonymous
- syntax:lambda argument :expression
  - lambdax:x^3(x->arg,x^3->exp)
- note:lambda takes 0 or more arguments but only one exp
  - ex:lambda x,y:x+y

In [398]:
```
##cube
result=lambda x:x**3
result(4)
```

Out[398]: 64

In [410]:
```
#sq
result=lambda x:x**2
result(4)
```

Out[410]: 16

In [411]:
```
list(map(lambda x:x**3,[1,2,4]))
```

Out[411]: [1, 8, 64]

In [412]:
```
even=lambda x:x%2==0
result=list(filter(even,range(20,40)))
print(result)
```

```
[20, 22, 24, 26, 28, 30, 32, 34, 36, 38]
```

In [ ]:
```

```