Scientific Computing with Python

Richard Bahin, Mathematics/Science, Full Sail University

Introduction

Scientific computing not only requires a deep knowledge of mathematics and the subject of application, but a strong skill in computer science and computer programming. In this section, we will look at how Python can be used for numerical modelling and visualization. We will explore some python packages such Numpy and Scipy for numerical computation , matplolib for graphing and visualization, and pandas for data representation.

A) Python programming language

1) Python variables and comments

Python comments:

# is used for a single line comment in python

‘’’ ‘’’ is used for multiple lines comment in python

Example:

```
# hello, I am a single line comment
‘’’
  hello,
  I am a multiple line comment
‘’’
```

Python variables:

Python does not explicitly define a variable type

```
score=9   # define a variable of type integer named score which value is 9
score=9.0   # define a variable of type float named score which value is 9
name=”USA” # define a variable of type string named name which value is USA
```

Python built-in types are :

int (integer), ***float* , *complex*** (complex or imaginary number) , ***str*** (string), ***bool***(boolean) ***dict***(dictionary), ***tuple* , *list***.

```
x1 = 1  # type: int
x2=int(1) # same as above using the int() constructor
y1 = 1.0  # type: float
y2=float(1.0)  # same as above using the float() constructor
c1=2+3j # type:complex (complex number or imaginary number)
c2=complex(2+3j)  # same as above using the complex() constructor

b1 = True  # type: bool
b2=bool(True) # same as above using the bool() constructor

s1 = "hello"  # type : str  (string)
s2=str("hello") # same as above using the str() constructor
s3 = u"mystring"  # type: str
t = b"test"  # type: bytes

# For collections, the name of the type is capitalized, and the
# name of the type inside the collection is in brackets.
lst1 = [1,2,3]  # type: List[int]
lst2=list([1,3,6]) # type: list[int] as above using list() constructor
```

```
y = {6, 7}  # type: Set[int]
y = set([6, 7])  # type: set[int] same as above using the set() constructor
# For mappings, we need the types of both keys and values.
d1 ={'name':"paul",'age':15}  # type: Dict[str, float]
d2=dict(name="Paul",age=15)  #same as above using the dict() constructor
d3=dict(name=str("Paul"),age=int(15) ) #same as above using the set() constructor with
                                              keys and values types
# For tuples, we specify the types of all the elements.
u1 = (3, "yes", 7.5)  # type: Tuple[int, str, float]
u2=tuple([10,"hello",3.14]) #same as above using the tuple() constructor
u3=tuple([int(10),str("hello"),float(3.14)])  # same  as above but specified types

# For textual data, use Text.
# This is `unicode` in Python 2 and `str` in Python 3.
x = ["string", u"unicode"]  # type: List[Text]
```

2) Python I/O , functions and modules

❖ Python uses the **input( )** for data input, and **print( )** for data output

```
name= input("what is your name")
  print("your name is " + name)
```

```
num1= input("Enter your first number:\n")
num2= input("Enter your second number:\n")
# convert from string to a number float
num1=float(num1)
num2=float(num2)
print("The sum of " + num1 + "and " + num2 + "is" + sum(num1, num2) )
print("\n")  # new line
print("The product of {0} and{1} is {2} ".format(num1,num2,num1*num2)  )
print("The difference of %d and %d is  %d "  %(num1,num2,num1-num2)  )
```

A print format could be : *print(" {0) , {1 }, {2} ".format(value1, value2, value3))*
Note that the function **input()** returns a string. So when used with numbers, its return which is a string needs to be casted into an **int** or **float** as follows:
num1=input("enter a number")
num1=int(num1)   or simply
num1=int (input("enter a number"))

❖ Python functions start with the word "**def** " and the function name followed by a colon " **:** " as follows

```
def sum(a,b):
      return a+b
```

```
def isGreater(x,y):
     return x>y
```

That is, the structure of a Python function is:

```
def function_name(parameter1,parameter2, parameter3,….) :
     statement(s)
     return return_value
```

Where the parameters can be any object including functions.
Notice the absence of semi-colon (;) after each statement in contrast to C++.
Notice the use of '#' for comment in python

- **Functions of more than one input or output** :
  Python functions can take multiple input arguments and return multiple output values.
  Syntax :

```
def   function_name(param1, param2, param3,….) :
statement(s)
return value1,value2,value3,….
```

Example:

```
import math
def circle(angle, radius):
      x=radius*cos(angle)
      y=radius*sin(angle)
  return x , y

  x,y=circle(90,2)   #  returns  x=0, y=2
```

- **Lambda Function**
  Lambda functions (anonymous functions) are small inline like functions created using the keyword **lambda** on a single line statement.The syntax is:

```
function_name = lambda  arg1 , arg2 , arg3 ,…. :     expression
```

Example:

```
g=lambda x , y : x + y +x*y
g(2,3)     # will return 2+3 + 2*3=11
```

- **variable number of arguments in function**
  Python uses the asterisk * to define a variable number of arguments in a function.

Structure:

```
def func_name( other_var , * args):
```

Example1:

```
def sumSquare(i,*val):
        sum=i
        for v in val:
            sum=sum + v*v
        return sum
```

sumSquare(6,1,2,3) will return 6 + 1*1 + 2*2 + 3*3=20

Example2 : with array or list

```
 from array import array
 def average4(num,*value):
    v=0
    for x in value:
       v+=x
    return v/num

 if __name__=="__main__":

   n=array('f',[12,6,4,10])  #   create an array of float
   print(average4(4,*n))    # notice  the * on n (*n) when using array or list
```

❖ Modules

In Python , a module is a file of functions to be loaded in a program by the statement:

```
import module_name
from  module_name  import  func1, func2,.....
import  module_name   as  something
```

Example :

```
 import math   # import the math library
 from math import cos, log, sin  # import some specific function from the math module
 import math  as M
```

3) Conditionals

Python **"if "** statement construct:

```
if expression:
   statement
```

Example:

```
x=4
if x>0:   x=x-1
```

Python "**if … else** " statement construct:

```
if expression:
   statement(s)
   else:
   statement(s)
```

```
Example:
 x=4
 if x>0:
    x=x-1
 else:
    x=x+1
 print(x)
```

will print: 3

Python "**elif…** ": same as c++ **else if** statement

statement construct

```
 Example:
 if x>1:
    x=x-1
elif x<1:
    x=x+1
else :
    x=1
```

```
if expression:
   statement(s)
elif expression:
   statement(s)
```

the operators of the conditional are:
<, > , >= , =< , ==, != , and , or ,not
The C++ equivalent of **and** , **or** , **not** are **&& , || , !**

4) **Loop control statements**
While loop , **break** and **continue** statement constructs :

Example:
```
x=10
while x>0:
    p=1/(x-10)
    print(p)
    print("\n")
    x=x+1
    if x==10: break
```

```
while expression:
        statement(s)
        if expression:
          break
        if expression:
          continue
```

for loop statement construct:

```
for index_variable  in range(value):
        statement(s)
```

Example1:
```
for i in range(5):
    print(i**2)
```

➔ will print : 0,1,4,9,16

Example2:
```
    for i in range(2,6,1):
     print(i**2)
```
➔ will print : 4,9,16,25

The range() function:
Note that **range(5)** is equivalent to C++ for loop (int i=0 ; i<5 ;i++) : 0 to 4 with 5 excluded
The function is **range(stop)** , where **stop** is the number of integer to generate starting from 0 to stop-1.
Example: range(4) gives 0,1,2,3
Also there is **range(start,stop,step) equivalent** to C++ **for** loop **(int i=start ; i <stop ; step++)**
stop, start and step must have integer value.

The random() function:

To use random numbers in Python, we will use the method random() after importing the library as:

```
import random
import random as rd
```

random.random( ) : generates numbers x in range [0,1)
random.randint(start,stop): generates random integer x in range [start,stop)
randomrange(start,stop,step): generates random integer x in range [start,stop]

example:
```
  import  random as rd
 x=rd.random()
 y=rd.randint(1,25)
 z=rd.randrange(1,25)
 print(("x={},y={},z={}").format(x,y,z))
```

it will print : x=0.5115963461166722, y=19 , z=12

5) Arrays

To use Python array we need to import the array library as follows:

```
from array import array
```

Array syntax:

```
value = array("type_code",[ x1,x2,….,xn])
```

Where type_code is listed in the table below :

Type code	C type	Python type
"b"	signed char	int
"B"	signed char	int
"h"	signed short	int
"H"	unsigned short	int
"i"	signed int	int
"I"	unsigned int	int
"f"	Float	float
"d"	Double	float
"l"	signed long	int
"L"	Unsigned	int

Example:

```
from array import array
# creates an array of integers:
arr1=array('i',[2,4,6,8,0])
# create an array of float:
arr2=array('f',[2.0,4.0,6.0])
for value in  arr2:
     print(value)                    #  ➔will print: 2.0 , 4.0 ,6.0
```

6) Python Strings

Strings are created in python using characters between double or single quote.
Example
str1='Richard'   # using single quote   .   str2="Bahin"   # using double quotes

Operator	Description	Examples
+	To concatenates 2 strings	str1+str2: RichardBahin
*	Repeats the same string	3*str2 : BahinBahinBahin
[]	Slice - Gives the character from the given index	str1[0] : ' R'
[:]	Range slice;gives the character from the given range Also use for partial copy	Str2[0:2] : 'Ba' str3=str2[:] : str2 = Bahin and str3= Bahin str2.append('r') : str2=Bahinr but str3=Bahin
In	Membership. return True if member	'a' **in** str : True
Not in	Membership. return True if not member	'a' **not in** str : False
=	Copy (clone) a string	str3=str2 : str2 = Bahin and str3= Bahin str2.append('r'): str2=Bahinr and str3=Bahinr

Escape characters:

"\n"	New line
"\t"	Tab
"\v"	Vertical tab

Some string functions:
Given str1="Richard"

Function	Description	Example
len()	Returns string length	len(str1): 7
find()	Returns char index	str1.find("a") : 4
replace(old,new)	To creates a new string	str1.replace("c","k"): Rickard str1.replace("r","D",5):RichaDd

7) **Lists**

List are sequence of arbitrary object which can be numbers or letters.
List are defined by two squared bracket on both ends with elements between double or single quote are separated by comas.
Lists are mutable, that is its elements and size can be changed.
List class and constructor name **: list / list( ).**

Example:
list1=["b,"a","h","i", "n"]
list2=[1,2,4,9]
list3=[1,2,"a","b"]
l=list([1,2,4])

or use the construct list( ) to create a list :   list4=list(["a", "b", "c"])
empty list: list4=list()

- Accessing value in a list:
  list1[0] : returns " b " ;
  list1[-2]: returns " i", counting from the right starting from -1 not zero.
- Updating a list:
  list1[2]="w"   : returns   " bawin"
  list1.**append**("e") : returns "bahine"   **append(object)** will add an object to the list.
- Deleting a list element":
  Use **del** to delete a list element at a given index:
  del list1[0] : will return   "ahin" deleting "a" at index 0
  del list1[0:2] **:** will delete "b","a" at index 0 and 1 and returns "hin"
  del list1 **:** will delete the entire list.

- **List operators:**
  Similar to the string operators previously listed:

Operators	results	description
len([9 , 1 ,1])	3	List length
[9, 1 ,1]+ [4,1 ,1]	[9 , 1 ,1, 4 , 1 ,1]	concatenation
2*[4, 1 ,1]	[4 , 1 ,1,4 , 1 ,1]	repetition
9 in [9, 1 ,1])	true	membership
for i in [4 , 1 ,1] : print i	4 1 1	iteration

❖ **List built-in functions and methods**
Functions:

Functions	Description	example
len(list)	Returns list length	len([4 , 1 ,1]): 3
cmp(list1,list2)	Compare the 2 lists elements	
max(list)	Return list max element	max([4 , 1 ,1]): 4
Min(list)	Return list min element	min([4 , 1 ,1]): 1
insert(index,object)	To add object at index	str.insert(1,"R"): RRichardX
list(tuple_sequence)	Convert a tuple into list	
Sort		

Methods:
Let   L=[4 , 1 ,1]

Methods	Description	examples
list.append(obj)	Add obj to list	L.append(9): [4 , 1 ,1,9]
list.count(obj)	Number of occurrence of obj	L.count(1) : 2
list.extend(seq)	Add the content of seq to list	M.extend(L): M=[4 , 1 ,1]
list.index(ojb)	Return index of obj in list	L.index("4") : 0
List.remove(obj)	Removes obj from list	L.remove("4") : L=[1,1]
List.reverse()		L.reverse() : [1 ,1,4]
List.sort([func])	Sort list based on func	

List are mutable objects, therefore the equality statement = if used does not create a new list but creates a reference to the list.
Example:
L1=["b","a","h","i","n"],
L2=L1   # L2 will be ["b","a","h","i","n"]
L2.append("X")   # both L1 and L2 will be ["b","a","h","i","n","X"]
That is any change made in L2 will be reflected in L1
So to create an independent list, use the slice operator [:], that is L2=L1[:]

8) Tuples

Tuples are sequence of immutable objects. Unlike lists, their element cannot be modified, and the parenthesis ( ) are used on both ends.
Class and constructor name **: tuple/ tuple( ).**

Example:
tup1=("b,"a","h","i", "n")
tup2=(1,2,4,9)
tup3=(1,2,"a","b")
or use the construct tuple( ) to create a tuple: tup4=tuple(["a","b"])
empty tuple: tup4=tuple( )

❖ Accessing value in a tuple:
tup1[0] : returns " b " ;
tup1 [-2]: returns " i", counting from the right starting from 1 not zero.

❖ Updating a tuple:
Tuples are immutable ,therefore cannot be updated once created

❖ Deleting a list element":
Tuples are immutable , therefore cannot be altered once created
But **del** tup1 : will delete the entire tup1.

❖ List operators:
Similar to the list operators previously listed:

❖ Tuples functions

Functions	Description	example
len(tup)	Returns tuple length	len([4 , 1 ,1]): 3
cmp(tup1, tup2)	Compare the 2 tuple elements	
max(tup)	Return list max element	
min(tup)	Return list min element	max([4 , 1 ,1]) : 4
tuple(list_sequence)	Convert a list into tuple	

9) Dictionary

A dictionary is an associative array where every key is associated to a value:
dict_name={ 'key' : value } , the key must be between 2 single quotes not double quotes
Class and constructor name **: dict / dict( ).**
example:
grade={'gorge': 89, 'jane': 100, 'katty': 95, 'polo': 90} or using the constructor **dict()**
grade=**dict**({'gorge': 89, 'jane': 100, 'katty': 95, 'polo': 90})
print("value is %s " % grade['jane']) # will print : value is 100
empty dictionary : grade={ } or grade=**dict**({ })

❖ **Accessing value in dictionary**
Values are accessed by using the square bracket with the key
```
>>print(" grade of jane is  %s" % grade['jane'])  will print :
>>grade of jane is  100
```

❖ **Updating dictionary**
Use a key/value pair to upgrade a dictionary as follows:
```
grade["paul"]=98
>>print(grade)
output>>    grade={'gorge': 89, 'jane': 100, 'katty': 95, 'polo': 90,'paul'=98}
```
❖ **Deleting dictionary elements**
Use **del** to remove one element from dictionary
```
>> del grade['paul']       # deletes paul
   print( grade)
>> grade={'gorge': 89, 'jane': 100, 'katty': 95, 'polo': 90 }
```

Can also be used to delete the entire dictionary: **del** grade

❖ **Dictionary methods**

```
dict={ 'a': 1 , 'b':2, 'c':3}
dict1={'d':4}
dict2=dict( {"name"= "paul", "age"=12 } ) # using the constructor dict( )
```

In the table below is a list of dictionary methods:

Methods	Description	Example
dict.clear()	Removes all dictionary elements	dict.clear() : will give dict={}
dict.copy()	Returns a shallow copy of a dictionary	dict1= dict.copy() : returns dict1={'a': 1, 'b': 2, 'c': 3}
dict.fromkeys(seq) or dict.fromkeys(seq,value)	Create a new dictionary with keys from seq and values set to value.	dict2=dict.fromkeys(dict) : gives dict2={‘a’:None,’b’:None,’c’:None } dict2=dict.fromkeys(dict,1) : gives dict2={‘a’:1,’b’:1,’c’:1}
dict.get(key,default=None)	For key key,returns value or default if not in dictionary	dict.get(“a”): gives 1
~~dict.has_key(key)~~ removed in Python 3. use ‘ in ‘ instead	Returns true if key in dict, or false	‘a’ in dict : gives true 8 in dict : gives false
dict.items()	Return a list of dict’s(key,value) tuple pair.	dict.items() : gives dict_items([('b', 2), ('a', 1), ('c', 3)])
dict.keys()	Return a list of dictionary keys	dict.keys(): gives dict_keys(['b', 'a', 'c'])
dict.update(dict2)	Adds dictionary dict2 key-values pair to dict	dict.update(dict1) : will give dict1={ ‘a’: 1 , ‘b’:2, ‘c’:3,’d’:4}
dict.values()	Returns list of dictionary dict’s values	dict.values() : gives dict_values([2, 1, 3])

10) Set

A set is a collection of objects of different types. It is similar to the set used in Mathematics. To create a set , use the constructor **set( ).**

S=set([1,2,3]) # in math it is $S = \{1,2,3\}$

S2=set([3,4,5]) # in math it is $S_1 = \{3,4,5\}$

S3=set() # empty set

S.intersection(*S1*) Returns the set of all items of *S* that are also in *S1* : $S \bigcap S_2 = \{3\}$

Example: S3=*S*.intersection(*S1*)
print(S3)
output>> { 3}

S.issubset(*S1*) Returns True if all items of *S* are also in *S1*; otherwise, returns False
S.issuperset(*S1*) Returns True if all items of *S1* are also in *S*; otherwise, returns False (like *S1*.issubset(*S*))
S.symmetric_difference(*S1*) Returns the set of all items that are in either *S* or *S1*, but not in both Sets $S - S_2 = \{1,2,4,5\}$
S.union(*S1*) Returns the set of all items that are in *S*, *S1*, or in both sets
mutating methods , $S \cup S_2 = \{1,2,3,4,5\}$
S.add(*x*) : Adds *x* as an item to *S*; no effect if *x* was already an item in *S*
S.clear( ): Removes all items from *S*, leaving *S* empty
S.discard(*x*): Removes *x* as an item of *S*; no effect if *x* was not an item of *S*
S.pop( ) : Removes and returns an arbitrary item of *S*
S.remove(*x*): Removes *x* as an item of *S*; raises a KeyError exception if *x* is not an element of S.

11) Python math operators

- **Python arithmetic operators**:

+ (addition), -(subtraction) * (multiplication), / (division),
(exponential) : 23 equivalent to $2^3=8$
// floor(division): 7//3 =2 returns the quotient
% (modulus operator or floor division): 7%3=1 returns the remainder

- **Comparison operators**

Operators	Description	Example
==	Logical equality	If a==b " print(a)
!=	Not equal	If a !=b print (b)
>	Greater than	If a>b print(a)
>=	Greater or equal to	If a>=b print(a)
<	Less than	If a<b print(b)
<=	Less or equal to	If a<b print(b)

- assignment operators

Operators	Description	Example
=	Assign value to variable	a=a -3
+=	Increment variable	a +=3 same as a=a+3
-=	Decrement variable	a -=3 same as a=a-3
*=	multiply	a *=2 same as a=2*a
/=	division	a /=2 same as a=a/2
%=	modulus	a %=2 same as a=a%
//=	Floor division	a //=2 same as a=a//2

- membership operators

Operator	Descriptions	example
in	Evaluates to true if element is member, otherwise false	A=[1,2,3] 2 in A : true
not in	Evaluates to true if element not member, otherwise false	A=[1,2,3] 4 in A : false

- identity operators

Operator	Descriptions	example

is	Evaluates to true if variables are the same.	P=5
		Q=5
		P is Q : true

is not	Evaluates to false if variables are not the same.	P=5
		Q=5
		P is not Q : false

12) File Management

Python has built-in function for operating files such opening , closing, reading from, and writing into a file. The process is defined as follows:

1. Open the file using open()
2. Do a read or write with
3. Close the file

Opening a file:

To open a file use the function ***open("filename", 'mode').***It returns a handle to the created file; **mode** defines how we want to use the file, write ('w'),read('r'),append('a') to the file.

The file can be open in text mode (.txt) or binary mode. Below are a summary of Python modes.

mode	description	examples
'w'	Opens a file for writing. Creates a new file if it does not exist or truncates the file if it exists.	f=open('file.txt', 'w')
'r'	Open a file for reading. (default)	f=open('file.txt' ,'r')
'a'	Opens for appending at the end of the file without truncating it. Creates a new file if it does not exist.	f=open('file.txt' ,'a')
'x'	Opens a file for exclusive creation. If the file already exists, the operation fails.	f=open('file.txt' ,'x')
't'	Opens in text mode. (default)	f=open('file.txt' ,'t')
'b'	Opens in binary mode.	f=open('file.txt' ,'b')
'+'	Opens a file for updating (reading and writing)	f=open('file.txt' ,'+')
'r+'	Open for reading/writing at file beginning	
'rb'	Open for read in binary format	
'rb+'	Opens for reading /writing in binary format	
'w+'	Opens for writing /reading	
'wb'	Opens for writing in binary format	
'wb+'	Opens for writing /reading in binary form	
'a+'	Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.	
'ab'	Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists.	
'ab+'	Opens a file for both appending and reading in binary format at end of file	

Closing a file:

To close a file , use the method ***close().***

The syntax is ***: fileobject.close()***

Example:

```
 f=open("myfile.txt",w)
 #do something
f.close()
```

Writing to a file:

To write to file , use the methods ***write()*** or ***writelines()*** after opening the file in write (‘w’), append(‘a’) mode. ***write()*** does not add a newline character (“\n”) to the end of the string

Example:

```
text="""
  Computer Programming is a science
  but it is also looked at as an art
  Its use has changed our world
  to be a beautiful place to live
  Richard Bahin
  """
  f=open("FILE_TEST.txt",'w')
  f.writelines(text)
  f.close()
```

Reading from a file:

To read from a file, use the method ***readline()*** or ***readlines()***

Example:

```
text="""
  Computer Programming is a science
  but it is also looked at as an art
  Its use has changed our world
  to be a beautiful place to live
  Richard Bahin
  """
  f=open("FILE_TEST.txt",'a')
  f.write(text)
  f.close()

  f=open("FILE_TEST.txt",'r')
  lines=f.readlines()
  for line in lines:
     print(line)
```

Opening a file using an application program

Using startfile() method to open a file using an application program like notepad if your file is a text file .txt. You will need import **os** (operating system) :

import os

the syntax is ***os.startfile(‘filename.txt’)*** or ***os.startfile(r ‘pathname’)***

Example:

```
import os
text="""
  Computer Programming is a science
  but it is also looked at as an art
  Its use has changed our world
  to be a beautiful place to live
  Richard Bahin
  """
  f=open("FILE_TEST.txt",'a')
  f.write(text)
  f.close()

  f=open("FILE_TEST.txt",'r')
  lines=f.readlines()
  for line in lines:
     print(line)
    os.startfile(‘FILE_TEST.txt’)    # will open file with notepad
#  or use  os.startfile(r 'C:\Users\rbahin\Documents\Python_Programming\FILE_TEST.txt')
```

13) Object Oriented Programming

❖ **Python class**

Python class starts with the keyword **class** and the class name with argument **object**. **__init__(self)** is the class constructor, a method of the class that is called every time an instance of the class is created. **__del__(self)** is the destructor of the class.

```
class   class_name :
__init__(self ):    # constructor
__del__(self):     #destructor
member attributes
member methods
```

```
class   class_name(object) :
__init__(self ):    # constructor
__del__(self) :    #destructor
member attributes
member methods
```

Member method syntax is : **def method_name(self):**

Example:

```
class student :
      __init__( self , lastName , firstName ):
            self.lastName=lastName         # class attribute member
            self.firstName= firstName       # class attribute member
            self.id=id                      # class attribute member
            self.grade=grade                # class attribute member
      def speakFrench(self ) :              # class method
            print("{0}  {1} speaks French \n".format(self.firstName,self.lastName ))
      def speakEnglish(self ):              # class method
            print("{0}  {1} speaks English \n".format(self.firstName,self.lastName ))
```

❖ **Creating an instance of the class and accessing class methods**

To create an instance of the class we do the following:

```
If __name__=="__main__":

      student1=student(Doe, John)
      student1.speakFrench()
      student2=student(Doe, Jane)
      student2.speakEnglish()
```

output:

```
      John Doe speaks French
      Jane Doe speaks English
```

Once the instance is created, it can access the class methods like the example below.

student1.speakFrench()
student2.speakEnglish()

❖ **Private methods and attributes**

In object oriented programming, private methods and attributes are those that can be called within the class but not outside the class.

The syntax of Python class method **is def methodName(self):**

Python methods and attributes are public by default.

To create a private method or attribute, a double underscore "__" is used preceding the method or attribute name .Below is the student class example.

```
class student :
    __init__(self,lastName,firstName):
            self.lastName=lastName          # class  public attribute member
            self.firstName= firstName       # class public  attribute member
            self.id=id                      # class public attribute member
            self.grade=grade
            self.__gender                   # class private attribute member
            self.__SSN                      # class private attribute member
    def speakFrench(self ) :            # class public method
        print("{0}  {1} speaks French \n".format(self.firstName,self.lastName ))
    def speakEnglish( self):            # public class method
        print("{0}  {1} speaks English \n".format(self.firstName,self.lastName ))
    def  __play(self ):                     # private method
          print("{0}  plays  soccer".format(self.firstName)
    def  __study(self ):                    # private method
           print("{0}  studies  math".format(self.firstName)
```

❖ **Static methods**

Static methods are class methods that cannot be accessed by an instance of the class, but by using the class name.

To create a static method in python, we use the **@staticMethod** decorator as follows:

```
class className :
@staticmethod
 def methodName(self ):
 def __init__(self):
```

To call the static method we do: **className.methodName()**

Example:

```
@staticmethod
    def dot(self,v1,v2):
        return v1.x*v2.x + v1.y*v2.y + v1.z*v2.z

class vector3(object) :
    __init__(self,x,y,z):
      self.x=x
      self.y=y
      self.z=z

if  __name__== ''__main__ '' :
    v1 = vector3(1,0,2)
    v2= vector3(2,9,3)
    result = vector3.dot(v1,v2)
    print("dot product is  {0}".format(result))
```

output : dot product is 8

❖ Proprieties

Private methods and attributes cannot be accessed outside the class definition.
One way to access private methods and attributes is to use property.
To write a property with read access we use the following the propriety decorator as follows:

```
@property
def  name(self):
      return self.__name
```

To write a property with write access we use the following the propriety decorator as follows:

```
@name.setter
 def name(self,value):
       self.__name=value

        Example:
        class   foo(object):
           def __init__(self):
                pass
           self.__age=10
           self.__id=1234AA
           @property
           def  age(self):
                 return self.__age
           @age.setter
            def age(self,value):
                 __age=value

   if  __name__==”__main__”:
          f=foo()
          f.age =90
          print("age is {}".format(f.age) )
```

output: age is 90

❖ Inheritance

Inheritance is the ability to create some new classes using some already created classes.
The former is called the derived classes, and the latter the based classes.
Python inheritance has the following syntax:

```
class A(object):
      def __init__(self,x):
           self.x=x
class B(A):
      def __init__(self,x,y):
        A.__init__(self,x)
        self.y=y

if __name__=="__main__":
  a=A(5)
  b=B(9,7)
  print('a.x={0} ,b.x={1} and b.y={2}'.format(a.x,b.x,b.y))
```

Example:

```
class employee(object):
     def __init__(self,name,age):
          self.name=name
          self.age=age
     def __str__(self):
         return("the employee is {0} with age {1}".format(self.name,self.age))

class programmer(employee):
     def __init__(self,name,age,skill):
          employee.__init__(self,name,age)
          self.skill=skill
     def __str__(self):
        return(employee.__str__(self)+ " his skill is {2} ".format(self.name,self.age,self.skill))

if __name__==”__main__”:
      p=programmer("Paul Hacker",45,"C++ programmer")
     print(p)
```

output: the employee is Paul Hacker with age 45 his skill is C++ programmer

Multiple inheritances are possible in Python.
Instead of using one based class, multiple classes are used to create the derive class.

multiple inheritances syntax :

```
class A(object):
        def __init__(self,x):
              self.x=x
        pass


class B(object):
        def __init__(self,y):
              self.y=y


class C(A,B):
        def __init__(self,x,y,z):
             A.__init__(self,x)
             B.__init__(self,y)
             self.z=z

if __name__=="__main__":

  a=A(5)
  b=B(9)
  c=C(1,2,3)
  print('c.x={0} ,c.x={1} and c.z={2}'.format(c.x,c.y,c.z))
```

Example :

```
class employee(object):
    def __init__(self,name,age):
        self.name=name
        self.age=age
    def __str__(self):
       return("the employee is {0} with age {1}".format(self.name,self.age))

class programmer(object):
    def __init__(self,skill,level):
        self.skill=skill
        self.level=level
    def __str__(self):
      return("Technical skill is {0}, mastery is {1} ".format(self.skill,self.level))


class mathematician(employee,programmer):
    def __init__(self,name,age,skill,level,expertise):
        employee.__init__(self,name,age)
        programmer.__init__(self,skill,level)
        self.expertise=expertise
    def __str__(self):
      return(employee.__str__(self) +" : " + programmer.__str__(self) + "an expert in {0}".format(self.expertise) )

if __name__=="__main__":
      p=mathematician("Paul Hacker",45,"C++","\n senior programmer", "Analysis")
      print(p)
```

```
output :  the employee is Paul Hacker with age 45 : Technical skill is C++, mastery is
          senior programmer an expert in Analysis
```

❖ **Operators overloading**

Python operators can be overloaded.
Below is the list of python operators that can be overloaded.

Operator	Method	
+	object.__add__(self,other)	Binary
-	object.__sub__(self,other)	Binary
*	object.__mul__(self,other)	Binary
//	object.__floordiv__(self, other)	Binary
%	object.__mod__(self, other)	Binary
/	object.__div__(self, other)	Binary
**	object.__pow__(self, other[, modulo])	Binary
<	object.__lt__(self, other)	Binary
<=	object.__le__(self, other)	Binary
>	object.__gt__(self, other)	Binary
>=	object.__ge__(self, other)	Binary
+=	object.__iadd__(self, other)	Binary
-=	object.__isub__(self, other)	Binary
*=	object.__imul__(self, other)	Binary
/=	object.__idiv__(self, other)	Binary
//=	object.__ifloordiv__(self, other)	Binary
**=	object.__ipow__(self, other[,m0dulo])	Binary
+	object.__pos__(self)	Unary
-	object.__neg__(self)	Unary
abs	object.__abs__(self)	Unary
~	object.__invert__(self)	Unary
!=	object.__ne__(self,other)	Binary
==	object.__eq__(self,other)	Binary
^	object.__xor__(self ,other)	Binary

Example:using type Python3 annotation(see section I for Python3 type annotation)

```
from typing import TypeVar
vector3=TypeVar("vector3")

class vector3(object):
    def __init__(self,x=0,y=0,z=0):
        self.x=x
        self.y=y
        self.z=z
    @staticmethod
    def dot(v1:vector3,v2:vector3)->float:
         return v1.x*v2.x+v1.y*v2.y+v1.z*v2.z

    def __str__(self):
        return("vector component are {0} ,{1} , {2}".format(self.x,self.y,self.z))

        # vector addition (u+v)
    def __add__(self,v):
        return vector3(self.x+ v.x , self.y + v.y , self.z + v.z)
     # vector difference (u-v)
    def __sub__(self,v):
        return vector3(self.x - v.x , self.y - v.y , self.z - v.z)

     # scalar vector multiplication (s*v)
    def mult(self,s:float)->vector3:
        return vector3(s*self.x,s*self.y,s*self.z)

     # vector dot product (u*v)
    def __mul__(self,v:type(vector3))->float:
        return self.x*v.x + self.y*v.y + self.z*v.z

    # cross product  of vectors (u^v)
    def __xor__(self,v):
        return vector3(self.y*v.z-self.z*v.y, self.z*v.x-self.x*v.z , self.x*v.y-self.y*v.x)

    def main():
    v1=vector3(1,0,2)
    v2=vector3(2,1,1)
    v3=v1+v2
    print("vector addition:{0}".format(v3))
    res=v1*v2
    print("dot product : {0}".format(res))
    res=v1^v2
    print("cross product : {0}".format(res))
    v4=v1.mult(10)
    print("scalar product by 10:{0}".format(v4))

if __name__=="__main__":

     main()
```

output :

```
vector addition: vector component are 3 ,1 , 3
dot product : 4
cross product : vector component are -2 ,3 , 1
scalar product by 10:vector component are 10 ,0 , 20
```

B) Numpy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- → a powerful N-dimensional array object
- → sophisticated (broadcasting) functions
- → tools for integrating C/C++ and Fortran code
- → useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined and this allows NumPy to seamlessly and speedily integrate with a wide variety of projects.

❖ Array creation

They are several ways to create a numpy array.The following syntaxes can be used after importing the numpy library as follows:

```
import numpy     or
import numpy as np
arr=np.array( [x1,x2,x3,…,xn] )
or
arr=np.array( [a1,a2,a3,…,an], dtype=value_type)
or
arr=np.array( [a1,a2,a3,…,an], value_type)
or for  a multidimensional array
arr=np.array(  [ [a1,a2,a3],  [b1,b2,b3], [c1,c2,c3] ] , dtype=value_type)

Where value_type=int,float,complex,double,int32,int64,float32,float64,
                       complex, str …..
```

Examples:

1) *in: r=np.array([1,2,3,4])*
   in:r
   out: array([1, 2, 3, 4])

2) *in: r=np.array([1,2,3,4],float)*
   in:r
   out: array([1., 2., 3., 4.])
3) *in: v=np.array([[1,2,0], [2,1,3], [0,0,1]], dtype=int)*
   in: v
   out:
```
array([[1, 2, 0],
       [2, 1, 3],
       [0, 0, 1]])
```

❖ Array creation special functions

The function ones():the function ***ones(shape,dtype=float,order='C')*** creates an array full of ones where **shape** is the array dimension , like (2,3) that is a 2x3 matrix default type is **floa**t, default array ordering (**order**) is the C/C++ ordering.
A Fortran ordering will be order='F'

Example:

```
In: Import numpy as np
In: np.ones((3,3))      # same as np.ones( (3,3) ,float, "c")
Out:
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
```

The function ones_like(): ones_like(a, dtype=none, order='K',subok ='True')

Creates an array of ones(1) with the same shape and type as a given array.
Example:

```
In: x=np.array([2,3,4])
In: x
Out: array([2, 3, 4])
In:  x=np.ones_like(x)
In:x
Out: array([1, 1, 1])
```

The function zeros(): the function ***zeros(shape,dtype=float,order='C')*** creates an array full of zeros where shape is the array dimension , like (2,3) that is a 2x3 matrix. default type is float, default array ordering is the C/C++ ordering. A Fortran ordering will be oder='F'

Example:

```
In: numpy.zeros((3,3))      # same as np.zeros( (3,3) ,float, "c")
Out[106]:
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
```

Return evenly spaced values within a given interval

The function zeros_like():zeros_like(a, dtype=none, order='K',subok ='True')
creates an array of zeros(0) with the same shape and type as a given array.

The function arange(): the arange(start, stop, step, dtype) function creates evenly spaced value within a specified interval **[start, stop)** where **start**=start number **stop**=stop number(excluded) **step**=spacing between values
example:

1) *# a 5-element array of int*

```
In:  numpy.arange(5)    # default function,start=0, stop=5,step=1
Out: array([0, 1, 2, 3, 4])
```

2) *# a 5-element array of float*

```
In:   numpy.arange(5.0)
Out: array([ 0.,  1.,  2.,  3.,  4.])
```

3)# element array in interval [1,5) with step size of 0.5

```
In: numpy.arange(1,5,0.5)
Out: array([ 1. ,  1.5,  2. ,  2.5,  3. ,  3.5,  4. ,  4.5])
```

The function empty( ): empty(shape,dtype=float,order''C')

Return a new array of given shape and type, without initializing entries.

Example:

```
In: e=numpy.empty((2,2),dtype=int)
Out:
    array([[0, 0],
           [2, 0]])
```

The function empty_like():empty_like(shape,dtype=float,order''C')

returns a new array with the same shape and type as a given array.

Example:

```
In :x=np.array([[1, 4],[ 2, 3]], float)
In :x
Out:
array([[ 1.,  4.],
       [ 2.,  3.]])
In: x=np.empty_like(x)
In :  x
Out:
array([[ 0.,  0.],
       [ 0.,  0.]])
```

The function eye(): eye(N,M=none,k, dtype='float')

returns a 2-D array with ones on the diagonal and zeros elsewhere.

N *: int ,*number of rows in the output.
M *: int, optional .* Number of columns in the output. If None, defaults to *N*.
k *: int, optional* index of the diagonal: 0 (the default) refers to the main diagonal, a positive value refers to an upper diagonal, and a negative value to a lower diagonal.
dtype *: data-type, optional .*Data-type of the returned array

Example 1:

```
In : e=np.eye(3,3,k=0,dtype=int)
In: e
Out:
array([[1, 0, 0],
       [0, 1, 0],
       [0, 0, 1]])
```

Example 2:

```
In: e=np.eye(4,k=1,dtype=int)
In:e
Out:
array([[0, 1, 0, 0],
       [0, 0, 1, 0],
       [0, 0, 0, 1],
       [0, 0, 0, 0]])
```

Example 3:

In: e=np.eye(4,k=-1,dtype=int)

```
In: e
Out:
array([[0, 0, 0, 0],
       [1, 0, 0, 0],
       [0, 1, 0, 0],
       [0, 0, 1, 0]])
```

The function **identity: identity(n, dtype=none) , dtype=float by default**
returns identity matrix
n:int , number of rows and columns

Example:

```
In:  np.identity(3)
Out:
 array([ [ 1.,  0.,  0.],
         [ 0.,  1.,  0.],
         [ 0.,  0.,  1.]])
```

The function **diag( ):** diag(v , k=0)
creates a diagonal matrix from a an array.
v : array_like
If `v` is a 2-D array, return a copy of its `k`-th diagonal.
If `v` is a 1-D array, return a 2-D array with `v` on the `k`-th diagonal.
k : int, optional
Diagonal in question. The default is 0. Use `k>0` for diagonals above the main diagonal, and `k<0` for diagonals below the main diagonal.

Example:

```
In : x = np.arange(9).reshape(3,3)
In :x
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
In : np.diag(x)
Out:array([0, 4, 8])
In : np.diag(x, k=1)
Out:array([1, 5])
In :np.diag(x, k=-1)
array([3, 7])
In : np.diag(np.diag(x))
Out: array([[0, 0, 0],
            [0, 4, 0],
            [0, 0, 8]])
In:n=np.diag([1,2,3])
In: n
Out: array([[1, 0, 0],
           [0, 2, 0],
           [0, 0, 3]]
```

The function **trace()** :   trace(a, offset=0, axis1=0, axis2=1, dtype=None, out=None)
Sum along diagonals.
a : array_like (2D). Input array, from which the diagonals are taken.
offset : int, optional.Offset of the diagonal from the main diagonal.
Can be both positive and negative. Defaults to 0.
axis1, axis2 : int, optional.Axes to be used as the first and second axis of the
2-D sub-arrays from which the diagonals should be taken.
Defaults are the first two axes of `a`.

Example:

```
 M=np.matrix([[2,3,4],[4,0,1], [1,2,5]])
print(M)
out :
[[2 3 4]
 [4 0 1]
 [1 2 5]]
np.trace(M)
Out: 7
```

The function **triu(): triu(m, k=0) . Returns a copy of a matrix with the elements below the `k`-th diagonal zeroed.**
The function tril(m,k=0)   **Returns a copy of a matrix with the elements above the `k`-th diagonal zeroed.**

Example:

```
T=np.triu([ [1,2,3],[3,5,7],[6,7,9] ] )
print(T)
[[1 2 3]
 [0 5 7]
 [0 0 9]]
```

The function full():
The function full_like():
The function linspace():
linspace(start,stop,num=50,endpoint=true,restep=false,dtype=none)
creates evenly spaced number over the interval [start, stop].
num= number of samples**; endpoint=True/False** to include **stop**
retstep=True/False to return value of step size
Example:

1) *#array of 6 float between [2,3]*
   In:np.linspace(2.,3.,num=6)
   Out: array([2. , 2.2, 2.4, 2.6, 2.8, 3.])
2) *#array of 6 float between [2,3) , 3 is excluded for endpoint=False*
   In: np.linspace(2.,3.,num=5,endpoint=False)
   Out: array([2. , 2.2, 2.4, 2.6, 2.8])
3) *#array of 6 float between [2,3] , with step size returned as 0.25*
   In: np.linspace(2.,3.,num=5,retstep=True)
   Out: (array([2. , 2.25, 2.5 , 2.75, 3.]), 0.25)

The function logspace():

logspace(*start*, *stop*, *num=50*, *endpoint=True*, *base=10.0*, *dtype=None*)

returns numbers spaced evenly on a log scale.

In linear space, the sequence starts at base ** start (*base* to the power of *start*) and ends with base ** stop .

start : float
base ** start is the starting value of the sequence.
stop : float
base ** stop is the final value of the sequence, unless *endpoint* is False. In that case, num + 1 values are spaced over the interval in log-space, of which all but the last (a sequence of length num) are returned.
num : integer, optional
Number of samples to generate. Default is 50.
endpoint : boolean, optional
If true, *stop* is the last sample. Otherwise, it is not included. Default is True.
base : float, optional
The base of the log space. The step size between the elements in ln(samples) / ln(base) (or log_base(samples)) is uniform. Default is 10.0.
dtype : dtype
The type of the output array. If dtype is not give

Example:

1) *Array numbers with default base 10*
   Start=100=10^2 stop=1000=10^3 at (2,3)
   Int: np.logspace(2,3)
   Out: array([100. , 104.81131342, 109.8541142 , ..., 910.29817799, 954.09547635, 1000.])

2) *Array numbers with default base 2*
   Start=4=2^2 stop=8=2^3 at (2,3)
   np.logspace(2,3,num=6,base=2)
   array([4.,4.59479342,5.27803164,6.06286627,6.96440451,8.])

The function meshgrid(x1, x2,..., xn ,):

Make N-D coordinate arrays for vectorized evaluations of N-D scalar/vector fields over N-D grids, given one-dimensional coordinate arrays x1, x2,..., xn
parameters:
x1, x2,..., xn *: array_like* .1-D arrays representing the coordinates of a grid.
indexing *: {'xy', 'ij'}, optional*
Cartesian ('xy', default) or matrix ('ij') indexing of output. See Notes for more details.
New in version 1.7.0.
sparse *: bool, optional.* If True a sparse grid is returned in order to conserve memory.
Default is False. *New in numpy version 1.7.0.*
copy *: bool, optional*
If False, a view into the original arrays are returned in order to conserve memory. Default is True. Please note that sparse=False, copy=False will likely return non-contiguous arrays. Furthermore, more than one element of a broadcast array may refer to a single memory location. If you need to write to the arrays, make copies first.
Returns :
X1, X2,..., XN *: ndarray*
For vectors *x1*, *x2*,..., 'xn' with lengths Ni=len(xi) , return (N1, N2, N3,...Nn) shaped arrays if indexing='ij' or (N2, N1, N3,...Nn) shaped arrays if indexing='xy' with the elements of *xi* repeated to fill the matrix along the first dimension for *x1*, the second for *x2* and so on.

Example 1:

```
In :x=np.arange(1,5,1)
In :y=np.arange(1,5,1)
In :X,Y=np.meshgrid(x,y)
In :X
Out:
array([[1, 2, 3, 4],
       [1, 2, 3, 4],
       [1, 2, 3, 4],
       [1, 2, 3, 4]])
In :Y
Out:
array([[1, 1, 1, 1],
       [2, 2, 2, 2],
       [3, 3, 3, 3],
       [4, 4, 4, 4]])
```

meshgrid ( )is very useful to evaluate functions on a grid.

Example2:

```
    x=np.linspace(0,6,num=3)
    y=np.linspace(0,8,num=3)
    X,Y=np.meshgrid(x,y)
    Z=X**2 + Y**2 -4
```

The function fromfunction(): fromfunction(function,shape,kwargs)**

Constructs an array by executing a function over each coordinate.

Example1:

```
  In :  np.fromfunction( lambda x,y :x**2+ y**2,(3,3))
 Out:   array([[ 0.,  1.,  4.],
               [ 1.,  2.,  5.],
               [ 4.,  5.,  8.]])
```

Example 2:

```
   In : def g(x,y):return 10*x+y
   In : z=np.fromfunction( g,(3,3) )
  Out:
  array([[  0.,   1.,   2. ],
         [ 10.,  11.,  12.],
         [ 20.,  21.,  22.]])
```

The function **where( ): where(condition [,x,y])**

returns elements, either from *x* or *y*, depending on *condition*.

Example 1:

```
In : np.reshape(x,(3,3))
Out: array([[ 0.,  1.,  2.],
            [ 3.,  4.,  5.],
            [ 6.,  7.,  8.]])
In : np.where(x>5)
Out: (array([6, 7, 8], dtype=int64),)
In :np.where( (x>=4) &  (x<=7) )
Out: (array([4, 5, 6, 7], dtype=int64)
```

❖ Printing array

When you print an array, NumPy displays it in a similar way to nested lists.
One-dimensional arrays are then printed as rows, bidimensionals as matrices and tridimensionals as lists of matrices.

Example1: 1D array

```
In :x=np.linspace(1,8,num=6,dtype=int)
In : print(x)
Out: [1 2 3 5 6 8]
```

Example 2: 2D array

```
m=np.arange(16).reshape(4,4)
print(m)
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
```

Example 3: 3D array , 2 matrices of order 3x4

```
In :R=np.arange(24).reshape(2,3,4)
In :print(R)
Out :[ [  [ 0  1  2  3]
          [ 4  5  6  7]
          [ 8  9 10 11] ]
       [  [12 13 14 15]
          [16 17 18 19]
          [20 21 22 23] ]
       ]
```

❖ Array mathematics

Arithmetic operators on arrays apply *elementwise*. A new array is created and filled with the result.

Example

```
a=np.array([1.,2.,3.],dtype=float)
b=np.array([4,0,3],float)
a+b
Out : array([ 5.,  2.,  6.])
a*b
Out : array([ 4.,  0.,  9.])
a-b
Out : array([-3.,  2.,  0.])
np.exp(a)
Out : array([  2.71828183,   7.3890561 ,  20.08553692])
a**2
Out : array([ 1.,  4.,  9.])
```

❖ Numpy elementwise functions

Most mathematical functions are provided in Numpy .These functions have been vectorized
for elementwise evaluation of arrays. Below are a list of some of the functions:
numpy.cos, numpy.sin, numpy.arctan, numpy.arccos, numpy.arcsin, numpy.sqrt, numpy.exp,
numpy.log, numpy.add, numpy.subtract, numpy.multiply, numpy.divide, numpy.power, numpy.sign, numpy.abs, numpy.round etc…

Example1:
u =array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
np.power(u,2)
Out: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

Example2
u =array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
np.round(np.sqrt(u),3)
Out: [0.   , 1.   , 1.414, 1.732, 2.   , 2.236, 2.449, 2.646,2.828, 3.   ]

A custom function applied on Numpy array will trigger an error message.
To build a custom vectorized function that will perform an elementwise evaluation on array,
Numpy provides a function *numpy.vectorize().It takes a non-vectorized function and returns it as a vectorized function.*
Example:

```
def funct(x):
   return 2*x+5

funct=np.vectorize(funct)
u=arange(6)
Out: array([0, 1, 2, 3, 4, 5])
v=funct(u)
Out: array([ 5,  7,  9, 11, 13, 15])
```

❖ **Numpy aggregate functions**

Aggregate functions take an array as input and return a single scalar as output.
The table below gives a list of Numpy aggregate functions.

`np.mean`	The average of all values in the array.
`np.std`	Standard deviation.
`np.var`	Variance.
`np.sum`	Sum of all elements.
`np.prod`	Product of all elements.
`np.cumsum`	Cumulative sum of all elements.
`np.cumprod`	Cumulative product of all elements.
`np.min`, `np.max`	The minimum / maximum value in an array.
`np.argmin`, `np.argmax`	The index of the minimum / maximum value in an array.
`np.all`	Return True if all elements in the argument array are nonzero.
`np.any`	Return True if any of the elements in the argument array is nonzero.

```
Example:
      m=np.array([[1,3,5],[0,2,4],[3,3,6] ])
    print(m)
    Out: [[1 3 5]
          [0 2 4]
          [3 3 6]]
    # compute the sum on each column
    w=np.sum(m,axis=0)
    Out: array([ 4, 8,  15 ])
    # compute the sum on each row
    np.sum(m,axis=1)
    Out: array([ 9,  6, 12])
    # compute the sum in the entire array(matrix)
    np.sum(m)
    Out: 27
```

❖ Array indexing, slicing

Numpy arrays can be indexed using the standard Python X[i] where X is the array and i is the index.A basic slicing has the following format X[start: end: step]; by default step=1 making X[start: end: step=1] and X[start: end] equivalent.

```
Example:
x=np.arange(10)
Out: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(x[2:9:2])
Out:[2 4 6 8]
```

For negative index, assuming X[i:j:k] where i , j , and k are respectively the start ,end ,and step indices, then i and j will be interpreted as i + n and j + n with n=number of elements in X

Negative step k makes its step backward towards smaller indices.

```
Example1:
  arr=np.array( [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] )
  x=arr[-2:10]  # same as  x[-2+10:10]=x[8:10]
  Out: array([8, 9])
Example2:
arr=np.array( [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] )
  x=arr[-2:10]  # same as  x[-2+10:10]=x[8:10]
  Out: array([8, 9])
 X=arr[-2:5:-3]  #  same as x[-2+10:10]=x[8:5:-3]  3 step backward from start=8 to end=5
Out: array([8])
X=arr[-3:2:-2]  #  same as x[-3+10:10]=x[7:2:-2]  2 step backward from start=7 to end=2
Out: array([7, 5, 3])
```

Array can also be sliced by using the following syntaxes :

X[i:] same as X[i::] meaning slice X starting from index i going forward

X[:i] same as X[::i] meaning slice X starting from index i going backward

X[:] same as X[::] meaning keep the original array

Example:

```
x =array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
x[3:]
Out: array([3, 4, 5, 6, 7, 8, 9])
x[:3]
Out: array([0, 1, 2])
x[3:7]
Out: array([3, 4, 5, 6])

x[::]
Out: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
x[:]
Out: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

- **Selecting matrix or higher order array value, row and column**

  Given a matrix M of order $n \times m$,with i the row index , j the column index, its Python definition will be **M[0:row_index , 0 : column_index]**

  Selecting the matrix value at entry (i , j):
  we get the value of a matrix at the i th row and j th column by typing **M[i,j]**.

  Row selection:
  To select a matrix row we type **M[row_index , :]** , where $0 \le row\_index < n$

  Column selection:
  To select a matrix column we type **M[: , column_index]** , where $0 \le column\_index < m$

  Example:

```
#creates the 3 by 3 matrix
M=np.matrix([ [1,0,1],[1,2,3],[2,5,4]])
Print(M)
Out:
[[1, 0, 1],
  [1, 2, 3],
  [2, 5, 4]])
#get the value at entry (3,3) that is (2,2) in python, since index starts from 0 not 1 in
Python
M[2,2]
Out: 4
#get the first row of the matrix
print(M[0,:])
Out:
 [[1 0 1]]
#get the first column of the matrix
print(M[:,0])
Out: [[1]
       [1]
       [2]]

#printing the matrix last column
print(M[:,2])
[[1]
 [3]
 [4]]
```

- Adding value to 1D array and 2D array
  numpy.append(arr, values, axis=None) appends values to the end of an array.
  arr and values must be array_like
  Example1:
  ```
   v=np.array([1,2,3,4])
   np.append(v,5)
  Out: array([1, 2, 3, 4, 5]
  np.append(v,[6,7,8])
  Out: array([1, 2, 3, 4, 6, 7, 8])
  ```
  Example2
  ```
   m=np.array([[1,2,3],[4,0,1]])
   print(m)
   out :[[1 2 3]
          [4 0 1]]
  m=np.append(m,[[7,7,7]],axis=0)   # append 7,7,7 to the end of each column of m
  Out:  array([[1, 2, 3],
               [4, 0, 1],
               [7, 7, 7]])

  m=np.append(m,[[0],[4],[8]],axis=0)   # append 0,4,8 to the end of each row of m
  Out: array([[1, 2, 3,0],
              [4, 0, 1,4],
              [7, 7, 7,8]])
  ```

- Vector(1D array) and matrix(2D array) stacking
  Stacking is done by using the following functions:
  numpy.vstack(tup): Stack arrays in sequence vertically (row wise)
  example:
  ```
  a = np.array([1, 2, 3])
  b = np.array([2, 3, 4])
  np.vstack((a,b))
  out: array([[1, 2, 3],
              [2, 3, 4]])
  a = np.array([[1], [2], [3]])
  b = np.array([[2], [3], [4]])
  np.vstack((a,b))
   out: array([  [1],
                 [2],
                 [3],
                 [2],
                 [3],
                 [4]])
  ```

  numpy.hstack(tup): Stack arrays in sequence horizontally (column wise)
  example:
  ```
  a = np.array((1,2,3))
  b = np.array((2,3,4))
  np.hstack((a,b))        # will output:      array([1, 2, 3, 2, 3, 4])
  a = np.array([[1],[2],[3]])
  b = np.array([[2],[3],[4]])
  np.hstack((a,b))         #   will output  :  array([ [1, 2],
                                                       [2, 3],
                                                       [3, 4]])
  ```

numpy.column_stack(tup): Stack 1-D arrays as columns into a 2-D array.

Example:

```
a = np.array((1,2,3))
b = np.array((2,3,4))
np.column_stack((a,b))
out: array([[1, 2],
            [2, 3],
            [3, 4]])
```

row_stack(): stack arrays in sequence vertically (row wise)

example:

```
a = np.array([1, 2, 3])
b = np.array([2, 3, 4])
np.row_stack((a,b))
  out: array([[1, 2, 3],
              [2, 3, 4]])

a = np.array([[1], [2], [3]])
b = np.array([[2], [3], [4]])
np.row_stack((a,b))
out: array([[1],
            [2],
            [3],
            [2],
            [3],
            [4]])
```

numpy.insert(arr,obj,values,axis=None): Insert values along the given axis before the given indices.

Example1: with 1D array

```
import numpy as np
n=np.array([1,2,3])
n=np.insert(n,2,10) # insert  10 at index=2  will output :  array([ 1,  2, 10,  3])
```

Example2 : with 2D array without **axis** specified

```
import numpy as np
m=np.array([ [1,2],[3,4] ])
Out[:array([[1, 2],
            [3, 4]])
m=np.insert(m,2,5)  # will  insert 5 at index=2  flattening the 2D array to 1D :  array([0, 2, 5, 3, 4])
```

Example 3: with 2D array with **axis** specified

```
import numpy as np
m=np.array([ [1,2],[3,4] ])  # will output :   array([[1, 2],
                                                       [3, 4]])
m=np.insert(m,1,5,axis=0) : inserts 5 at index 1 in each column :     array([[1, 2],
                                                                              [5, 5],
                                                                              [3, 4]]
```

- **Numpy Linear Algebra Matrix and Vector**

Numpy provides a specialized type of array ***numpy.matrix()*** to represent matrix and vector found in Linear Algebra. Matrix and vector algebra can be evaluated on this type of array.

Row vector representation *:*

v=np.matrix([[1,2,3]])   # a 1x3 row vector   or

v=np.matrix("1,2,3")   # this will print   [[1 2 3]]

Column vector representation:

```
v=np.matrix([ [1],[2],[3]])   or
v=np.matrix([ [1],
              [2],
              [3]
                ])  # a 3x1 column vector  or
```

v=np.matrix("1 ;2 ;3") # notice the use of semi-colons to represent the end of rows

This will print
```
[[1]
 [2]
 [3]]
```

Matrix creation: the use of **numpy.matrix( )** or **numpy.mat()** creates matrix of any order.

Example:

In: *mat=np.matrix([[1,0,1],[1,2,3],[2,5,4]])* *or*
mat=np.matrix("1,0,1; 1,2,3; 2,5,4")

In :print(mat)

Out :
```
[[1 0 1]
 [1 2 3]
 [2 5 4]]
```

The semi colons **(;)** are used to represent the end of the matrix rows.
Below is a list of some Numpy matrix functions.

NumPy Function	Description
`np.dot`	Matrix multiplication (dot product) between two given arrays representing vectors, arrays, or tensors.
`np.inner`	Scalar multiplication (inner product) between two arrays representing vectors.
`np.cross`	The cross product between two arrays that represent vectors.
`np.tensordot`	Dot product along specified axes of multidimensional arrays.
`np.outer`	Outer product (tensor product of vectors) between two arrays representing vectors.
`np.kron`	Kronecker product (tensor product of matrices) between arrays representing matrices and higher-dimensional arrays.
`np.einsum`	Evaluates Einstein's summation convention for multidimensional arrays.

Matrix vector multiplication:

Given a matrix $M = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 2 & 3 \\ 2 & 5 & 4 \end{pmatrix}$ and a column vector $\vec{v} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$, we want to compute

$\vec{u} = M \cdot \vec{v} = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 2 & 3 \\ 2 & 5 & 4 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 4 \\ 14 \\ 24 \end{pmatrix}$. Python uses the * operator or the function **dot()** for matrix-vector multiplication.

Example:
In Python we will do:

```
#create the 3 by 3 matrix M
In: M=np.matrix([ [1,0,1],[1,2,3],[2,5,4]])
#create the 3 by 1 column vector v
v=np.matrix([ [1],[2],[3] ])
#multiply using the * operator
u=M*v
out:   [ [ 4]
         [14]
         [24] ]
# or use the function dot()
u=M.dot(v)
```

Matrix-matrix multiplication:
Python uses the * **operator** or the function ***numpy.dot()*** for matrix-matrix multiplication.

Given $M=\begin{pmatrix}1&0&1\\1&2&3\\2&5&4\end{pmatrix}$ and $P=\begin{pmatrix}1&0&1\\1&1&3\\0&0&1\end{pmatrix}$, we want to compute

$$Q=M\cdot P=\begin{pmatrix}1&0&1\\1&2&3\\2&5&4\end{pmatrix}\cdot\begin{pmatrix}1&0&1\\1&1&3\\0&0&1\end{pmatrix}=\begin{pmatrix}1&0&2\\3&2&10\\7&5&21\end{pmatrix}$$

Example:

```
#create the 3 by 3 matrix M
In: M=np.matrix([ [1,0,1],[1,2,3],[2,5,4]])
#create the 3 by 3 matrix P
In: P=np.matrix([ [1,0,1],[1,1,3],[0,0,1]])
#multiply M and P using the * operator
Q=M*P
#or multiply M and P using the dot() function as follows Q=M.dot(P)
Out:
 [  [ 1,  0,  2],
    [ 3,  2, 10],
    [ 7,  5, 21] ]
```

Matrix addition: Python uses the **+ operator** or the ***numpy.add(m1,m2)*** function to add 2 matrice**s**

Given $M=\begin{pmatrix}1&0&1\\1&2&3\\2&5&4\end{pmatrix}$ and $P=\begin{pmatrix}1&0&1\\1&1&3\\0&0&1\end{pmatrix}$, we want to compute

$$S=M+P=\begin{pmatrix}1&0&1\\1&2&3\\2&5&4\end{pmatrix}S=M+P=\begin{pmatrix}1&0&1\\1&2&3\\2&5&4\end{pmatrix}+\begin{pmatrix}1&0&1\\1&1&3\\0&0&1\end{pmatrix}=\begin{pmatrix}2&0&2\\2&3&6\\2&5&5\end{pmatrix}$$

Example:
#create the 3 by 3 matrix M
In: M=np.matrix([[1,0,1],[1,2,3],[2,5,4]])
#create the 3 by 3 matrix P
In: P=np.matrix([[1,0,1],[1,1,3],[0,0,1]])
*#multiply M and P using the * operator*
Q=M+P
#or add M and P using the ***numpy.add()*** *function*
Q=np.add(M,P)

Out:

```
    [[2, 0, 2],
    [2, 3, 6],
    [2, 5, 5]])
```

To subtract 2 matrices , Python uses the **– operator** or **numpy.subtract(m1,m2)**

Vector inner dot product: use numpy.inner(m1,m2)

Given $\vec{a} = (1,2,3)$ and $\vec{b} = (1,4,2)$ we want to compute the dot(inner) product of $\vec{a} \bullet \vec{b} = (1,2,3) \bullet (1,4,2) = 1+8+6 = 15$

Example:
a=np.array([1,2,3])
b=np.array([1,4,2])
np.inner(a,b)
Out: 15
Or
a=np.matrix([1,2,3])
b=np.matrix([1,4,2])
np.inner(a,b)
Out: [[15]]

Vector outer dot product: use numpy.outer(m1,m2)

Given $\vec{a} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$ and $\vec{b} = \begin{pmatrix} 1 \\ 4 \\ 2 \end{pmatrix}$ we want to compute the outer dot product ,that is

$$m = \vec{a} \bullet \vec{b}^T = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \begin{pmatrix} 1 & 4 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 4 & 2 \\ 2 & 8 & 4 \\ 3 & 12 & 6 \end{pmatrix} \quad \text{where } \vec{b}^T \textit{ is the transpose of } \vec{b}$$

Example:
a=np.matrix([[1],[2],[3]])
b=np.matrix([[1],[4],[2]])
m=np.outer(a,b)
Out:

```
array([[ 1,  4,  2],
       [ 2,  8,  4],
       [ 3, 12,  6]])  # or  *  and the transpose of b , b.T
```

M=a*b.T

Vector addition: to add two vectors(row vectors in this case), Python uses the **+ operator** or the **numpy.add(v1,v2)** function **.**

Given $\vec{a} = (1,2,3)$ and $\vec{b} = (1,4,2)$ we want to compute the sum $\vec{c} = \vec{a} + \vec{b} = (1,2,3) + (1,4,2) = (2,6,5)$

Example:

```
a=np.matrix([1,2,3])
b=np.matrix([1,4,2])
c=a+b
#or use np.add()
c=np.add(a,b)
Out :  [[2, 6, 5]]
```

Trace of a matrix :

Given $M = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 2 & 3 \\ 2 & 5 & 4 \end{pmatrix}$, we want to compute trace(M)=1+2+4=7.

Python uses **numpy.trace(matrix)** for this purpose

Example:

```
M=np.matrix([ [1,0,1],[1,2,3],[2,5,4]])
np.trace(M)
Out: 7
```

Matrix transpose:

Given $M = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 2 & 3 \\ 2 & 5 & 4 \end{pmatrix}$, we want to compute $M^T = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 2 & 3 \\ 2 & 5 & 4 \end{pmatrix}^T = \begin{pmatrix} 1 & 1 & 2 \\ 0 & 2 & 5 \\ 1 & 3 & 4 \end{pmatrix}$.

Python uses **numpy.transpose(mat)** or numpy.**T** for this purpose.

Example:

```
M=np.matrix([ [1,0,1],[1,2,3],[2,5,4]])
m=np.transpose(M)
#or using  numpy.T
m=M.T
Out: [[1, 1, 2],
      [0, 2, 5],
      [1, 3, 4]]
```

Matrix determinant: To compute the determinant of a matrix, Python uses ***numpy.linalg.det(mat)*** or simply ***linalg.det()*** function

example:

```
M=np.matrix([ [1,0,1],[1,2,3],[2,5,4]])
det=linalg.det(M)
print(det)
out : -6
```

Matrix inverse: The inverse of a matrix is computed by using the function ***linalg.inv(mat)*** **.**

Given $M = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 4 \\ 5 & 6 & 0 \end{pmatrix}$ we want to compute $M^{-1} = \begin{pmatrix} -24 & 18 & 5 \\ 20 & -15 & -4 \\ -5 & 4 & 1 \end{pmatrix}$

Example :

```
M=np.matrix([ [1,2,3],[0,1,4],[5,6,0 ] ])
print(M)
[[1 2 3]
 [0 1 4]
 [5 6 0]]
det=linalg.det(M)
print(det)
Out:1

inverse=linalg.inv(M)
print(inverse)
Out :
[[-24.  18.   5.]
 [ 20. -15.  -4.]
 [ -5.   4.    1.]]
```

Solving system of linear equations:

Given the following system of linear equation $\begin{cases} x+y+z=3 \\ 2x+y+z=4 \\ x-y+2z=5 \end{cases}$**,** we want to find x, y and z. We need first the matrix expression of the equation in the form $A\cdot\vec{x}=\vec{b}$**,** that is :

$$\begin{pmatrix} 1 & 1 & 1 \\ 2 & 1 & 1 \\ 1 & -1 & 2 \end{pmatrix}\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 3 \\ 4 \\ 5 \end{pmatrix} \quad \text{where} \quad A = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 1 & 1 \\ 1 & -1 & 2 \end{pmatrix}, \ \vec{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \ \text{and} \ \vec{b} = \begin{pmatrix} 3 \\ 4 \\ 5 \end{pmatrix}$$

The solution will be x=1 , y=0, z=2 or solution vector $\vec{x} = \begin{pmatrix} 1 \\ 0 \\ 2 \end{pmatrix}$

Python uses **linalg.solve(mat,vec)** to solve a system of linear equations

Example: solving the previous equation using linalg.solve()

```
A=np.matrix([ [1,1,1],[2,1,1], [1,-1,2] ])
b=matrix([ [3],[4],[5]] )
x_vector=linalg.solve(A,b)
print(x_vector)
[[ 1.]
  [0.]
 [ 2.]]
```

Finding eigen values and eigen vectors:

Given a matrix M, we want to find some scalar λ and vector $\vec{v}$ such that $M\cdot\vec{v}=\lambda\vec{v}$. λ is called the eigen value and its corresponding vector $\vec{v}$ is called the eigen vector. We solve the characteristic equation $\det(M-\lambda\cdot I)=0$ to find λ.

Then the value of λ will be used to compute $\vec{v}$ using $M\cdot\vec{v}=\lambda\vec{v}$ or better $(M-\lambda I)\vec{v}=\vec{o}$

Let's solve one example.

EX: Find the eigen value and corresponding eigen vectors of $M=\begin{pmatrix}-5 & 2\\ 2 & -2\end{pmatrix}$

Answer:

The characteristic equation is $\det(M-\lambda\cdot I)=0$ or $\begin{vmatrix}-5-\lambda & 2\\ 2 & -2-\lambda\end{vmatrix}=0$ ➔

$(-5-\lambda)(-2-\lambda)-4=\lambda^2+7\lambda+6=0$ ➔ eigen value $\lambda_1=-1$ *and* $\lambda_2=-6$

For $\lambda_1=-1$, we have $(M-\lambda_1 I)\vec{v}_1=\vec{o}$ ➔ $(M+I)\vec{v}_1=\vec{o}$ ➔

$\begin{pmatrix}-4 & 2\\ 2 & -1\end{pmatrix}\begin{pmatrix}x\\ y\end{pmatrix}=\vec{o}$,

$2x-y=0$ ➔ $y=2x$ ➔ $\vec{v}_1=\begin{pmatrix}x\\ y\end{pmatrix}=\begin{pmatrix}x\\ 2x\end{pmatrix}=x\begin{pmatrix}1\\ 2\end{pmatrix}$,

So we pick $\vec{v}_1=\begin{pmatrix}1\\ 2\end{pmatrix}$ for eigen vector when $\lambda_1=-1$ finally normalize it as

$\hat{v}_1=\begin{pmatrix}\frac{1}{\sqrt{5}}\\ \frac{2}{\sqrt{5}}\end{pmatrix}\approx\begin{pmatrix}0.4472136\\ 0.89442719\end{pmatrix}$

For $\lambda_2=-6$, we have $(M-\lambda_2 I)\vec{v}_2=\vec{o}$ ➔ $(M+I)\vec{v}_2=\vec{o}$ ➔ $\begin{pmatrix}1 & 2\\ 2 & 4\end{pmatrix}\begin{pmatrix}x\\ y\end{pmatrix}=\vec{o}$,

$x+2y=0$ ➔ $\vec{v}_2=\begin{pmatrix}x\\ y\end{pmatrix}=\begin{pmatrix}-2y\\ y\end{pmatrix}=y\begin{pmatrix}-2\\ 1\end{pmatrix}$. So we pick $\vec{v}_2=\begin{pmatrix}-2\\ 1\end{pmatrix}$ for eigen vector when $\lambda_2=-6$; finally normalize it as $\hat{v}_2=\begin{pmatrix}\frac{-2}{\sqrt{5}}\\ \frac{1}{\sqrt{5}}\end{pmatrix}\approx\begin{pmatrix}-0.89442719\\ 0.4472136\end{pmatrix}$.

To compute the eigen value and vectors in Python we use:

numpy.linalg.eigvals() : computes eigenvalues of a non-symmetric array(matrix).

Example:

```
#create the 2 by 2 matrix
M=np.matrix([ [-5,2] ,[2,-2]])
print(M)
Out :
 [[-5  2]
 [ 2 -2]]
#get the eigen values of M
eigen_val=linalg.eigvals(M)
print(eigen_val)
Out:
[-6.  -1.]
```

numpy.linalg.eig(): computes eigen values and right eigen vectors.
For instance in ***w,v = linalg.eig(M),*** *where M is a square matrix,* ***w*** *is the list of eigen value and* **v** *the corresponding eigen vectors*

Example *:*
#create the 2 by 2 matrix M

```
M=np.matrix([ [-5,2] ,[2,-2]])
print(M)
[[-5  2]
 [ 2 -2]]
# get both the list of eigen values  and eigen vectors  matrix using   linalg.eig( )
eigen_value , eigen_vector =linalg.eig(M)
print(eigen_value)
[-6. -1.]
# from  the eigen vectors  matrix, get v1 as the first column vector
v1=eigen_vector[:,0]
print(v1)


[[-0.89442719]
 [ 0.4472136 ]]
# from  the eigen vectors  matrix, get v2 as the second column vector
v2=eigen_vector[:,1]
print(v2)
[[-0.4472136 ]
 [-0.89442719]]
```

Note that either [[-0.4472136] , [-0.89442719]] or [[0.4472136] , [0.89442719]] is solution as eigen vector of M (they are opposite).

numpy.linalg.*eigh():* computes eigenvalues and eigenvectors of a symmetric or Hermitian (conjugate symmetric) array (matrix).

numpy.linalg.*eigvalsh():* computes eigenvalues of a symmetric or Hermitian (conjugate symmetric) array.

Important remark:

The aforementioned functions return normalized (unit “length”) eigenvectors, such that the column v[:,i] is the eigenvector corresponding to the eigenvalue w[i].

C) Visualization with Matplotlib

In scientific computing , data visualization is a very important tool .Data represented on a graph is more expressive in the sense that it helps to communicate information more efficiently and clearly. Here we will look at Python main visualization tool Matplotlib. To plot using Matplotlib, we need to import two packages, **numpy** and **matplotlib** as follows:

import numpy as np
import matplotlib.pyplot as plt

To plot we use the function ***plot(*args, **kwargs)***
Each function below is legal .

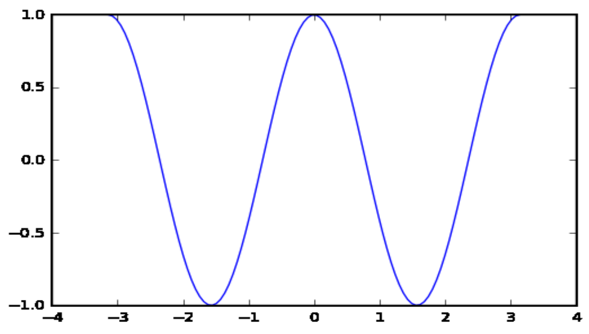
plot(x, y) # plot x and y using default line style and color
plot(x, y, 'bo') # plot x and y using blue(**b**) circle markers (**o**) as **'bo'**
plot(x, y, color='green', label='line 1', linewidth=2) # with custom color,label,
# line thickness
plot(x, y, color='green', linestyle='dashed', marker='o', markerfacecolor='blue', markersize=12).
plot(y) # plot y using x as index array 0..N-1
plot(y, 'r+') # ditto, but with red plusses

- **Curve plotting**

  - **Single curve plotting**

    To plot a curve in Matplolib we need to import the following libraries :
    import numpy as np
    import matplotlib.pyplot as plt
    Example: to draw f(x)=cos(2x) from –π to π over two periods.

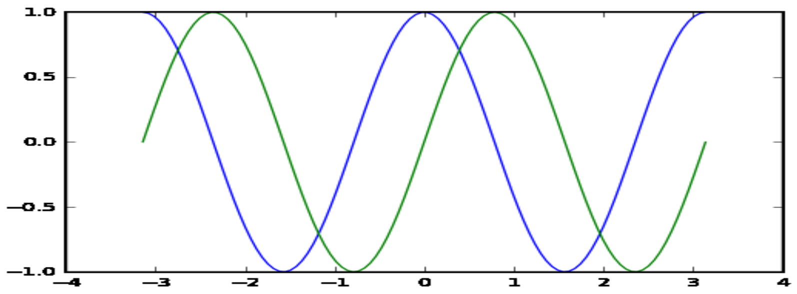
```
import numpy as np
import matplotlib.pyplot as plt
x=np.linspace(-np.pi,np.pi,100,dtype=float)
y=np.cos(2*x)
plt.plot(x,y)
plt.show()
```



❖ **Multiple curves plotting**

Example: *to draw $f(x)=\cos(2x)$ and $f(x)=\sin(2x)$ from $-\pi$ to π over two periods on the same graph.*

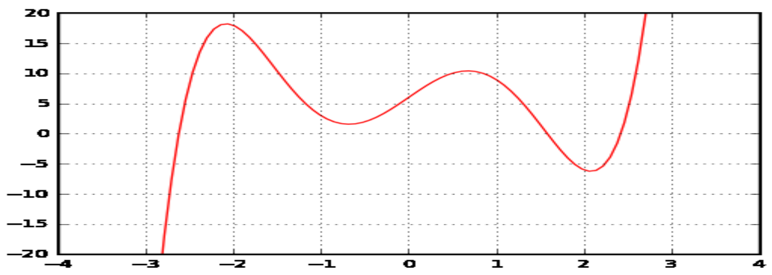
```
import numpy as np
import matplotlib.pyplot  as plt

x=np.linspace(-np.pi,np.pi,100,dtype=float)
y1=np.cos(2*x)
y2=np.sin(2*x)
plt.plot(x,y1)
plt.plot(x,y2)
plt.show()
```



❖ **Adding custom color , setting a Y-axis value limit and grid lines**

Example : to plot $f(x) = x^5 - 8x^3 + 10x + 6$ with a red color over the limit [-20,20] using **pyplot.ylim(ymin,ymax) , and pyplot.grid(True)**

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

def f(x):
    return x**5 - 8*x**3 + 10*x + 6

plt.ylim(-20,20)  #
plt.plot(x,f(x),color="red")     #  using  plot(x, y, color)  , color="red" or color="r"
plt.grid(True)
plt.show()
```



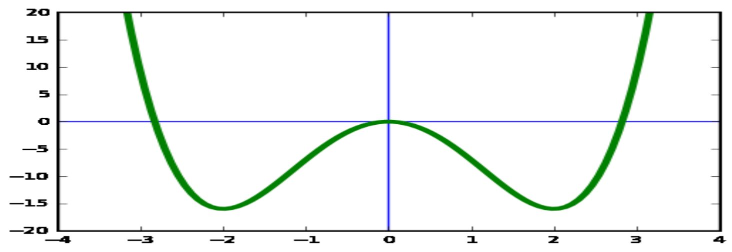
the custom colors are : **red(r),blue(b),green(g),cyan(c),magenta(m),black(k),yell**

❖ **Adding custom thickness to curve , vertical and horizontal axis**

To add a vertical and an horizontal axis to the curve we use the functions **pyplot.axvline()** and **pyplot.axhline().**

Example: to plot $f(x) = x^4 - 8x^2$ with linewidth=4 and x-axis and y-axis

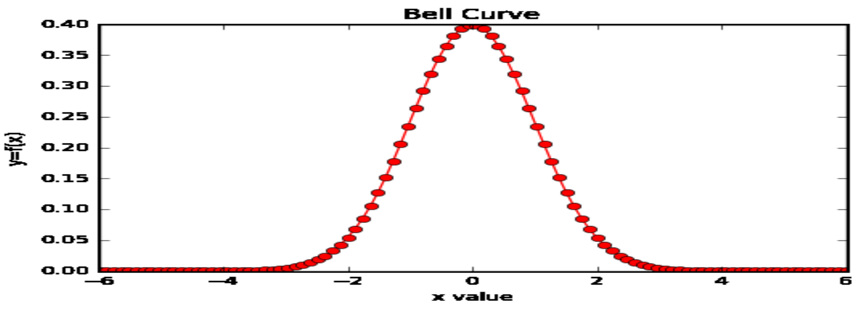
```
import numpy as np
import matplotlib.pyplot as plt
f=lambda x:  x**4 - 8*x**2
x=np.linspace(-4,4,100,dtype=float)
y=f(x)
plt.axvline(0,color="blue")
plt.axhline(0,color="blue")
plt.ylim(-20,20)
plt.plot(x,y,color="green",linewidth=4)  # with line thickness at 4
plt.show()
```



❖ **Adding labels , title and markers**

Example: plotting the Bell Curve $f(x) = \frac{1}{\sqrt{2\pi}}e^{-0.5x^2}$ **with labels, titles and markers**

```
  import numpy as np
  import Matplotlib.pyplot as plt
     x=np.linspace(-6,6,100)
 f= lambda x: (1/np.sqrt(2*np.pi))*np.e**(-0.5*x**2)
 plt.plot(x,f(x),'r-o')   # red o markers
 plt.xlabel("x value" )  # label on x value
 plt.ylabel("y=f(x)")      # label on y value
 plt.title("Bell Curve")   # adding title
 plt.show()
```



Matplotlib marker options:

Marker Code	Marker Displayed
+	Plus Sign
.	Dot
o	Circle
*	Star
p	Pentagon
s	Square
x	X Character
d	Diamond
h	Hexagon
^	Triangle

❖ Adding texts, annotations and picks

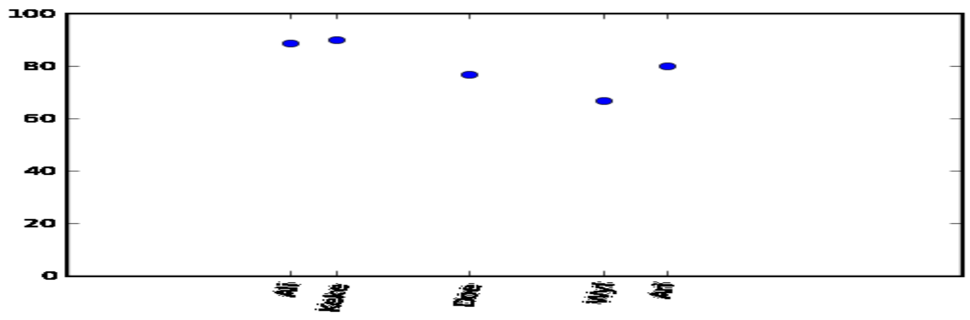
The **pyplot.text(x,y,text)** adds text around a figure at location x,y.
To add an annotation use **pyplot.annotate(text,xy,xytext,arrowprops)** where :
xy : the arrow head position in (x,y) format
xytext: text position in (x,y) format
arrowprops: arrow properties , a dictionary in the following format,
arrowprops=dict(facecolor='color',arrowhead=1,….) with keys listed below

`arrowprops` key	description
width	the width of the arrow in points
frac	the fraction of the arrow length occupied by the head
headwidth	the width of the base of the arrow head in points
shrink	move the tip and base some percent away from the annotated point and text
**kwargs	any key for `matplotlib.patches.Polygon`, e.g., `facecolor`

To add ticks to a plot, use:
pyplot.xticks(locs,labels,rotation) or **pyplot.yticks(locs,labels,rotation)** respectively on the x- and y-axis where **locs** is an array of tick locations, **labels** is an array of tick labels and
rotation is the labels orientation.
For instance: **xticks**([89,90,77,100,67],["Ali","Keke","Doe","Ari","Wyl"], rotation=90)

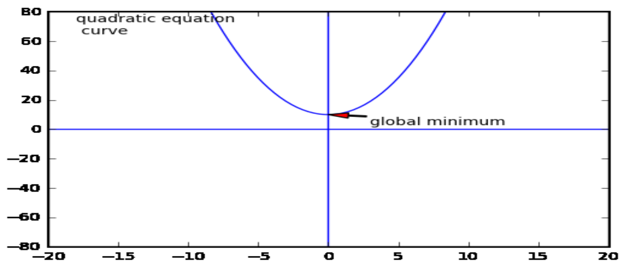
Example:example of ticks used

```
import matplotlib.pyplot as plt
x=[25,30, 45,67,60]
y=[89,90,77,80,67]
plt.xticks([25,30, 45,67,60],['Ali','Keke','Doe','Ari','Wyl'],rotation=80)
plt.plot(x,y,'o')
plt.xlim(0,100)
plt.ylim(0,100)
plt.show()
```



Example: text and annotation example

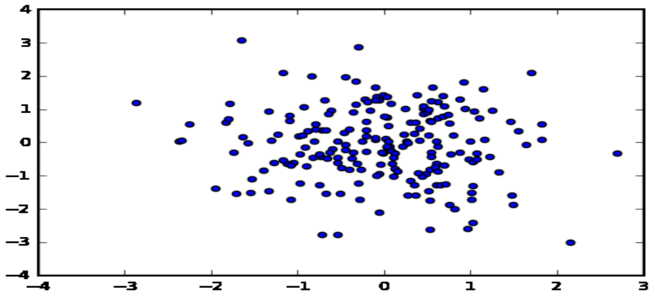
```
import numpy as np
import matplotlib.pyplot as plt
x=np.linspace(-20,20,100)
plt.xlim(-20,20)
plt.ylim(-80,80)
f=lambda x:(x)**2+ 10
plt.plot(x,f(x))
plt.axvline()
plt.axhline()
plt.text(-18,65,"quadratic equation\n curve")
plt.annotate("global minimum",xy=(0,10),xytext=(3,3),arrowprops=dict(facecolor='red',width=1,headwidth=5,shrink=1))
plt.show()
```



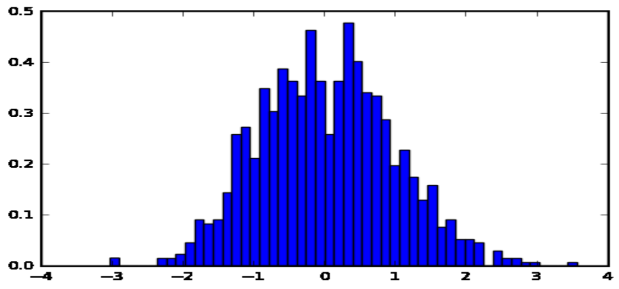
- Scatter plotting
  Matplotlib provides a function pyplot.scatter() for scatter plotting.

  Example: scatter plot pyplot.scatter() and numpy.random.randn()

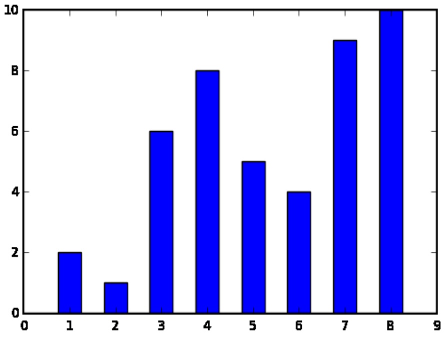
```
import numpy as np
import  matplotlib.pyplot as plt
x=np.random.randn(200)
y=np.random.randn(200)
plt.scatter(x,y)
plt.show()
```



- Histogram plotting
  hist(x, bins=10, range=None, normed=False, weights=None, cumulative=False, bottom=None, histtype='bar', align='mid', orientation='vertical', rwidth=None, log=False, color=None, label=None, stacked=False, hold=None, data=None, **kwargs)
  Example:
  import numpy as np
  import matplotlib.pyplot as plt
  x=np.random.normal(size=1000)
  plt.hist(x,bins=50,normed=True)
  plt.show()

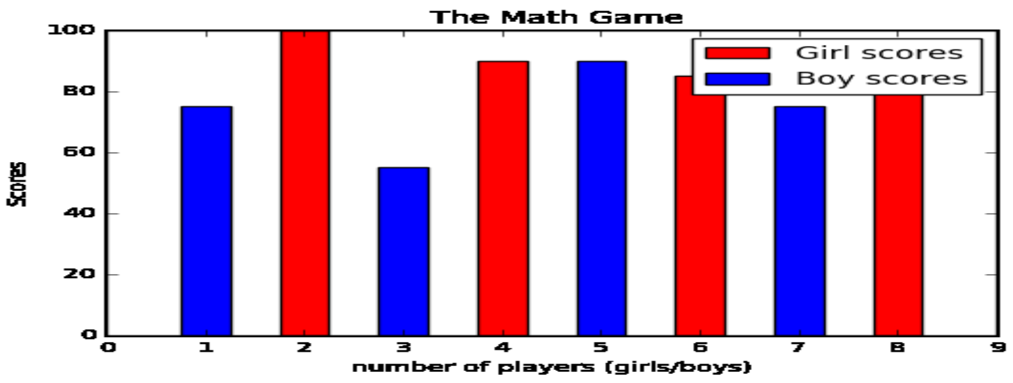


- Barchart plotting , legend
  Example :
  import numpy as np
  import matplotlib.pyplot as plt
  x=np.linspace(1,8,8,dtype=float)
  y=np.array([2.0, 1.0, 6.0, 8.0, 5.0, 4.0, 9.0, 10.0])
  plt.bar(x,y,width=0.5,align='center')
  plt.show()



Example 2: Barchart with legend

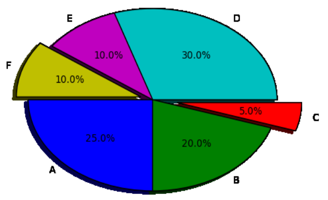
```
        import numpy as np
        import matplotlib.pyplot as plt
    x1=[ 2,4,6,8]
    y1=[100,90,85,95]
    x2=[1,3,5,7]
    y2=[75,55,90,75]
    plt.bar(x1,y1,width=0.5,align='center',label="Girl scores",color="red")   # legend
    label defined
    plt.bar(x2,y2,width=0.5,align='center',label="Boy scores",color="blue") # legend
    label defined
    plt.xlabel("number of players (girls/boys)")  # chart  x-axis label
    plt.ylabel("Scores")                           # chart  y-axis label
    plt.title(" The Math Game")
    plt.legend()                    # to display the legend from plt.bar(x,y,
    label=”legend_text”)
    plt.show()
```



- Pie chart plotting
  Use the pyplot.pie() to plot a pie chart.
   plot.pie(data, explode, labels, colors, shadow, labeldistance, startangle, explode,radius, center, frame)

  example:

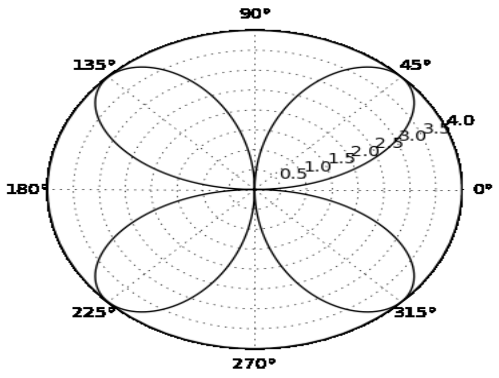
```
import numpy as np
import matplotlib.pyplot as plt
 data=[25,20,5,30,10,10]
labels=['A','B','C','D','E','F']
# explode the second and the sixth part, with shadow  , data percentage
plt.pie(data,labels=labels,explode=(0,0,0.2,0,0,0.1),shadow=True,startangle=180,autopct="
%1.1f%%")
plt.show()
```



- Polar coordinate plotting
  Using **pyplot.axes(polar=True)** for polar plot.

  Example 1: plotting $r(\theta) = 4\sin(2\theta)$, $0 \le \theta \le 2\pi$ in polar coordinate

```
 Import numpy as np
Import Matplotlib.pyplot as plt
  theta = np.linspace(0 , 2 * np.pi, 1024)
  plt.axes(polar = True)
  plt.plot(theta, np.abs(4.0 * np.sin(2 * theta)), c= 'k')
  plt.show()
```

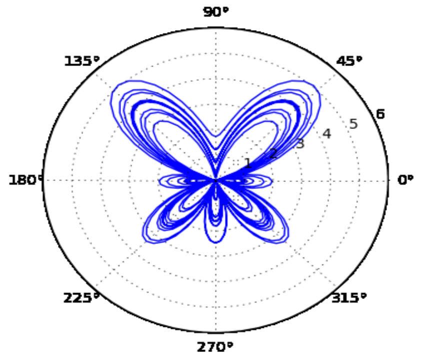


Example2: to draw butterfly equation ,

$$r(\theta) = e^{\sin\theta} - 2\cos(4\theta) + sin^5[\frac{1}{24}(2\theta - \pi)],\ \ 0 \le \theta \le 24\pi$$

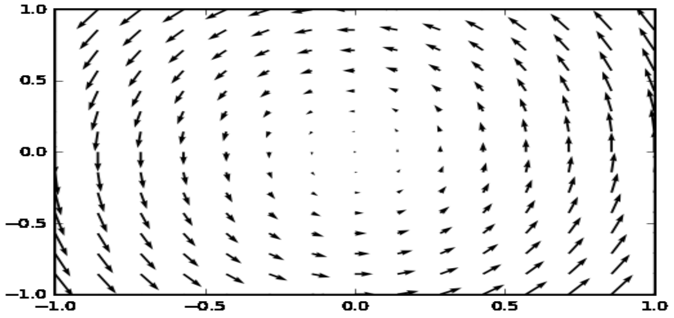
using **pyplot.polar(angle, radius)** function

```
 import numpy as np
 import matplotlib.pyplot as plt
 t=np.linspace(0,24.0*np.pi,num=1024,dtype=float)

def r(t):
   radius=np.e**np.sin(t)-2*np.cos(4*t)+np.sin( (1.0/24.0)*(2*t-np.pi))**5
   radius=np.abs(radius)  # radius should  be returned positive
   return radius

 plt.polar(t,r(t))
 plt.show()
```



- Quiver or vector plot
  Plotting vector is done using plotly.quiver() function.
  quiver(x,y,u,v,angle , scale ,color) where u and v are the parametric function expressed in terms of x,y which are the points coordinate, angle='xy' if wanted to be calibrated on the XY-axis.
  Example:

```
import numpy as np
import matplotlib.pyplot as plt
X=np.linspace(-1,1,num=15)
Y=np.linspace(-1,1,num=15)
x,y=np.meshgrid(X,Y)
u=-2*y
v= 3*x
plt.quiver(x,y,u,v)
plt.show()
```

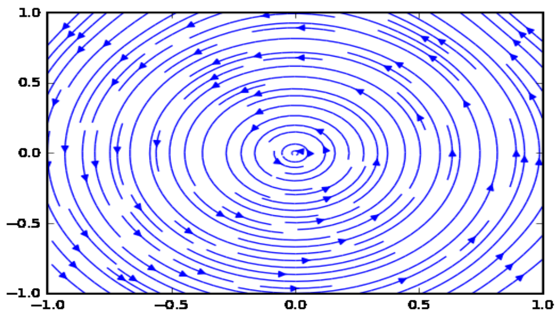


- Stream plot
  Plotting a stream (vector field) in Python is done using pyplot.streamplot() function
  streamplot(x,y,u,v,transform,linewidth,density,arrowsize,color).
  where x,y are data coordinates, u and v are the parametric function expressed in terms of x,y like
  in the following vector field:

$$\vec{F}(x,y) = u(x,y)\vec{i} + v(x,y)\vec{j} = -2y\vec{i} + 3y\vec{j} \quad \text{where} \quad u = -2y \quad and \quad v = 3x$$

  example:

```
import numpy as np
import matplotlib.pyplot as plt
X=np.linspace(-1,1,num=10)
Y=np.linspace(-1,1,num=10)
x,y=np.meshgrid(X,Y)
plt.xlim(-1,1)
plt.ylim(-1,1)
u=-2*y
v= 3*x
plt.streamplot(x,y,u,v,density=1.5,arrowsize=1.5)
plt.show()
```



- Subplot , Figure ,Axis, and Saving Plotting

  Matplotlib provides functionality to draw multiple plots on the same window screen(figure) using pyplot functions **figure( )** ,**axis()** and **subplot()** .

  A figure object is a GUI(Graphics User Interface) of the canvas where a plot is to be displayed.

  An axis object is the space defined by the x-axis and y-axis on the canvas where the plot is drawn.

  figure(num, figsize=(width,height) , facecolor)

  figsize: (optional) it defines the width and height of the canvas.

  num: string as a window title or number as the window identification number.

  facecolor: (optional) defines the figure background color.

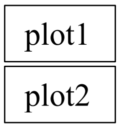
  e.g: figure(1,figuresize=(800,600),facecolor=(1,0,1)) or
  figure("hello window",figuresize=(800,600),facecolor=(1,0,1))

  subplot(numrows, numcols, fignum) :

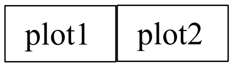
  numrows: number of rows; **numcols**: number of columns**; fignum:** number of figure to be plotted,

  fignum=numrows*numcols is the maximum figures obtained.

  So to create a figure of 2 plots in 2 rows and 1 column, we do :
  subplot(2,1,1), subplot(2,1,2), or subplot(211) , subplot(212).

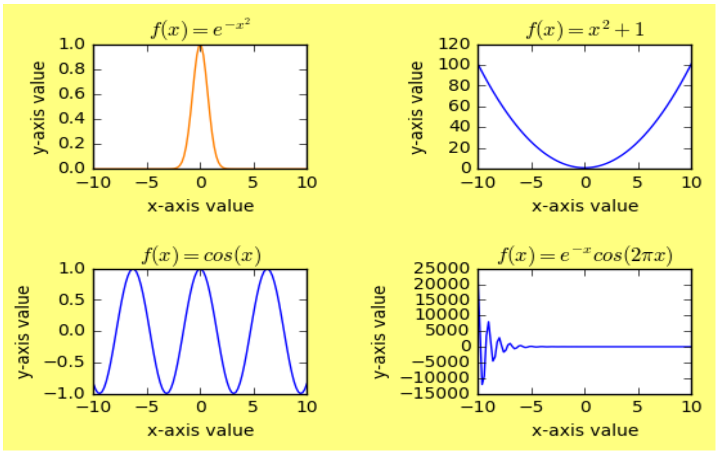


  To create a figure of 2 plots in 1 row and 2 columns, we do :
  subplot(1,2,1), subplot(1,2,2), or subplot(121) , subplot(122).



  Example: Creating 4 plots in 2 rows and 2 columns

```
import numpy as np
import matplotlib.pyplot as plt
x=np.linspace(-10.0,10.0,num=100)
def f1(x): return np.exp(-x**2)
def f2(x): return x**2+1
def f3(x): return np.cos(x)
def f4(x): return np.exp(-x) * np.cos(2*np.pi*x)

fig=plt.figure("An example of subplot",facecolor=(1,1,0.5))
plt.subplot(221)
plt.xlabel("x-axis value")
plt.ylabel("y-axis value")
plt.title(r"$f(x)=e^{-x^2} $")
plt.plot(x,f1(x),color=(1,0.5,0))

plt.subplot(222)
plt.xlabel("x-axis value")
plt.ylabel("y-axis value")
plt.title(r"$f(x)=x^2+1 $")
plt.plot(x,f2(x))

plt.subplot(223)
plt.xlabel("x-axis value")
plt.ylabel("y-axis value")
plt.title(r"$f(x)=cos(x)$")
plt.plot(x,f3(x))
plt.subplot(224)
plt.xlabel("x-axis value")
plt.ylabel("y-axis value")
plt.title(r"$f(x)=e^{-x}cos(2\pi{x}) $")
plt.plot(x,f4(x))
fig.subplots_adjust(hspace=.80) # to add space between the plots ,width and height
fig.subplots_adjust(wspace=.80)fi
plt.show()
```

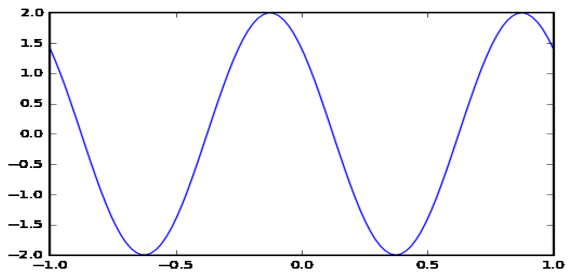


- Saving a plot with **pyplot.savefig()**

  To save a plot in Matplotlib, use the method **pyplot.savefig()** .
  savefig(“filename.ext”) , or savefig(“directory\filename.ext”) where
  ext (file extension) is png, pdf,csv

Example:

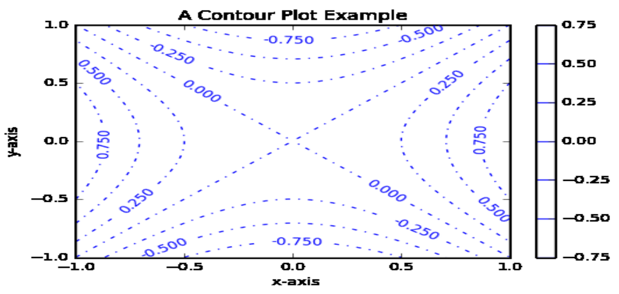
```
import numpy as np
    import matplotlib.pyplot as plt
    x=np.linspace(-1,1,num=100)
    def f(x): return 2*np.cos(2*np.pi*x + np.pi*0.25)
    plt.plot(x,f(x))
    # save file “cosine” as a pdf file
    plt.savefig("C:/Users/rbahin/Documents/Python_Programming/cosine.pdf")
    # save file “cosine” as a png file
    plt.savefig("C:/Users/rbahin/Documents/Python_Programming/cosine.png")
    plt.show()
```



- Contour plot
  A contour line(isoline) of a function of two variables is a curve along with the function has a constant
  value, $f(x,y)=c$.
  Example: a circle at (1,2) with radius 4 has as equation $(x-1)^2+(y-2)^2=4$ could written as
  $f(x,y)=4$ where $f(x,y)=(x-1)^2+(y-2)^2$ or $z=f(x,y)=(x-1)^2+(y-2)^2-4=0$
  To plot a contour use:
  pyplot.contour() or **pyplot.contourf()** for a filled contour.
  pyplot.contour(x,y,z,colors,level,linewidths,linestyles,antialiased) with optional key words :
  `level`: level number to draw, level=[1,2,3]
  `linewidths`: [`None` | number | tuple of numbers]
  `linestyles`: [`None` | 'solid' | 'dashed' | 'dashdot' | 'dotted']
  `antialiased`: [`True` | `False`]
  To add a contour label use **pyplot.*clabel()*.**
  A contour bar can be added by calling **pyplot.*cbar()*.**

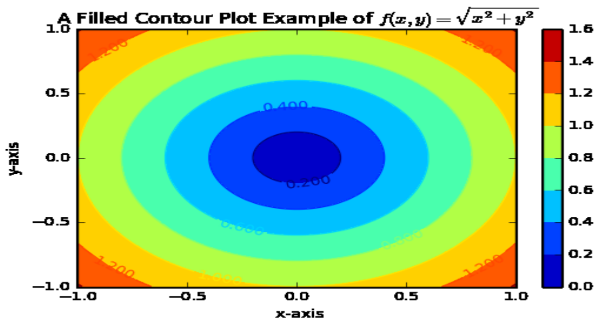
  Example 1: contour diagram of $z(x,y)=x^2-y^2$ with label and label

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

X=np.linspace(-1,1,num=50)
Y=np.linspace(-1,1,num=50)
x,y=np.meshgrid(X,Y)
z=x**2-y**2
cp=plt.contour(x,y,z,colors='b',linestyles='dashdot')
plt.colorbar(cp)
plt.clabel(cp)
plt.xlabel(" x-axis")
plt.ylabel("y-axis")
plt.title("A Contour Plot Example")
plt.show()
```



Example 2: Filled contour diagram of $z(x,y)=\sqrt{x^2+y^2}$ with label example

```
  import numpy as np
  import matplotlib.pyplot as plt
X=np.linspace(-3.0,3.0,100)
Y=np.linspace(-3.0,3.0,100)
x,y=np.meshgrid(X,Y)
z=np.sqrt(x**2 + y**2)
plt.figure()
cp=plt.contour(x,y,z) # create contour with label
plt.clabel(cp)
cf=plt.contourf(x,y,z) # create filled  contour with contour bar
plt.colorbar(cf)
plt.xlabel(" x-axis")
plt.ylabel("y-axis")
plt.title("A Filled Contour Plot Example of  $f(x, y)= \sqrt{x^2+y^2}$")
    plt.show()
```



- 3D plotting

To plot 3D surface , we need to import the following files in our code:

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D.

Then create a figure using **pyplot.figure()** or **pyplot.figure(figsize=(screen_width, screen_height))** if
the aspect ratio needs to be defined, that is fig=pyplot.figure(figsize=(80,20)) aspect ratio of 0.25.

Once the figure (fig) is created, we create or generate the axis using **pyplot.gca(projection='3d'**) .

The rest will be to generate your mesh grid data using **numpy.meshgrid().**

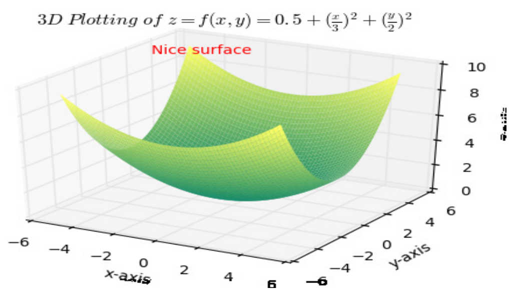
❖ ***Plotting surfaces in 3D***

plot_surface(x,y,z,rstride,cstride,color,vmin,vmax,cmap,linewidth,antialised)
cmap=cm.summer,linewidth=0,antialiased=False)
rstride: *Array row stride (step size), defaults to 10*
cstride: *Array column stride (step size), defaults to 10*
color: *Color of the surface patches*
vmin: *Minimum value to map*
vmax: *Maximum value to map*
cmap: *color map* , ***linewidth:*** *(0 for nice plot),* ***antialised:*** *True|False*

Also to add labels, titles, limits use Axes.set_label(), Axes.set_title(),Axes.set_xlimit

Example:

```
 import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D as axes
from matplotlib import cm  # to import color map

fig = plt.figure()
ax = fig.gca(projection='3d')
x_grid = np.linspace(-5,5,num = 60)
y_grid = np.linspace(-5,5,num=60)
x,y =np.meshgrid(x_grid, y_grid)  # get  the data

def z(x,y): return 0.5 + (x/3)**2 + (y/2)**2
ax.plot_surface(x,y,z(x,y),cstride=1,rstride=1,cmap=cm.summer,linewidth=0,antialiased=False)
ax.set_title("3D Plotting Example")
ax.set_xlabel("x-axis")
ax.set_ylabel("y-axis")
ax.set_zlabel("z-axis")
ax.text(0, 10, 10, "red", color='red')
#axes.set_
plt.show()
```

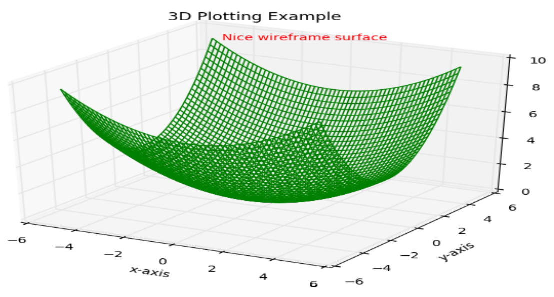


- *Plotting wireframes in 3D*

  To plot a wireframe , use pyplot.plot_wireframe() method.
  pyplot.plot_wireframe(X, Y, Z, rstride, cstride,color, *cmap,linewidth,antialised*)

  Example:

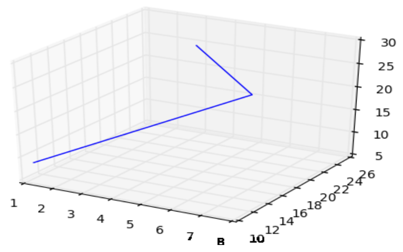
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D as axes
from  matplotlib import cm

def z(x,y): return  0.5 + (x/3)**2 + (y/2)**2

fig = plt.figure(figsize=(8,6))
ax=fig.gca(projection='3d')
X = np.linspace(-5,5,num = 60)
Y = np.linspace(-5,5,num=60)
x,y =np.meshgrid(X,Y)
ax.plot_wireframe(x,y,z(x,y),cstride=1,rstride=1,color='g',cmap=cm.summer,linewidt
h=1.1)
ax.set_title("3D Plotting Example")
ax.set_xlabel("x-axis")
ax.set_ylabel("y-axis")
ax.set_zlabel("z-axis")
ax.text(-4, 4, 10, "Nice wireframe surface", color='red')
plt.show()
```



❖ *Plotting lines and curves in 3D*
Use Axes3D.plot3D() or Axes3D.plot() method to plot a curves(parametric) and a line in 3D.

Example:line example

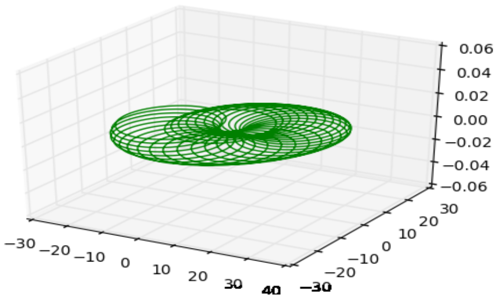
```
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
fig = plt.figure()
ax = plt.axes(projection='3d')
xline=np.array([1,8,3])
yline=np.array([11,12,25])
zline=np.array([8,29,25])
ax.plot3D(xline,yline,zline,'b-')
plt.show()
```



Example 2 :curve of parametric equation

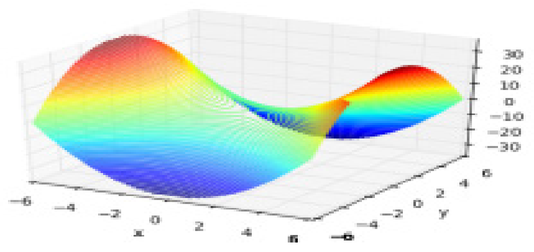
$$\begin{cases} x = 4\pi t + 4\pi\left[\cos(2\pi n t) - \cos(2\pi t)\right] \\ y = \cos(4\pi t) + \left[4\pi\sin(2\pi n t) - \sin(2\pi t)\right] \\ n = 30\ ,\ 0 \le t \le 1 \end{cases}$$

```
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.mplot3d import art3d as art
import matplotlib.pyplot as plt

PI=np.pi
t=np.linspace(0,1,num=2000)
n=30
x=4*PI*t + 4*PI*( np.cos(2*PI*n*t) - np.cos(2*PI*t))
y=np.cos(4*PI*t)+ 4*PI*(np.sin(2*PI*n*t) - np.sin(2*PI*t))
fig=plt.figure()
ax=fig.gca(projection='3d')
ax.plot(x,y,color='g')
```



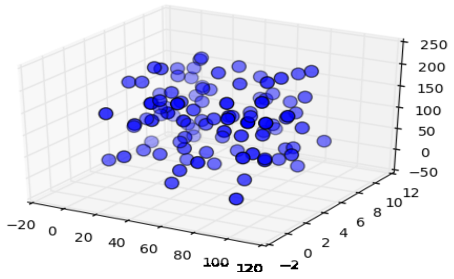
- ***Plotting contours in 3D***
  Use pyplot.contour() or pyplot.contour3D(). *For filled contour use* ***contourf3D() or contourf().***
  contour(X, Y, Z, strides,num,color,*args, **kwargs)

  Example:
```
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.mplot3d import art3d as art
import matplotlib.pyplot as plt

def f(x,y): return x**2-y**2

fig = plt.figure()
ax = plt.axes(projection='3d')
x = np.linspace(-6, 6, 30)
y = np.linspace(-6, 6, 30)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)
ax.contour3D(X, Y, Z,100,color='r')
ax.set_xlabel('x')
ax.set_ylabel('y')
plt.show()
```



- ***Plotting scatter plots in 3D***
  Use ***pyplot.scatter() or pyplot.scatter3D()*** *o draw a 3D scatter plot*
  scatter3D() (xs, ys, zs=0, zdir='z', s=20, c='b', depthshade=True, *args, **kwargs)
  xs,ys,zs: positions of data points**.**
  s:size of point^2 . default is 20
  c:*color* ***zdir****: direction to use, could be 'x', 'y', or 'z'.*
  depthshade: shade to give the appearance of depth, default=**True**

  Example:
```
import numpy as np
#from mpl_toolkits import mplot3d
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.mplot3d import art3d as art
import matplotlib.pyplot as plt

fig = plt.figure()
ax = plt.axes(projection='3d')
zdata =200* np.random.random(100)
xdata =100* np.random.random(100)
ydata = 10*np.random.random(100)
ax.scatter3D(xdata, ydata, zdata, c='b',depthshade=True, s=90,cmap='Greens');
plt.show()
```



❖ ***3D Surface Triangulation***

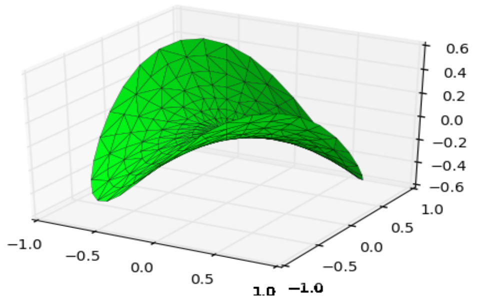
Use **Axes3D.plot_trisurf(*args, **kwargs)** method to triangulate a surface.

The arguments are:

x,y,z	Data values as 1D arrays
color	Color of the surface patches
cmap	A colormap for the surface patches.
norm	An instance of Normalize to map values to colors
vmin	Minimum value to map
vmax	Maximum value to map
shade	Whether to shade the facecolors

Example:

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np


n_radii = 8
n_angles = 36
# Make radii and angles spaces (radius r=0 omitted to eliminate duplication).
radii = np.linspace(0.125, 1.0, n_radii)
angles = np.linspace(0, 2*np.pi, n_angles, endpoint=False)

# Repeat all angles for each radius.
angles = np.repeat(angles[..., np.newaxis], n_radii, axis=1)

# Convert polar (radii, angles) coords to cartesian (x, y) coords.
# (0, 0) is manually added at this stage,  so there will be no duplicate
# points in the (x, y) plane.
x = np.append(0, (radii*np.cos(angles)).flatten())
y = np.append(0, (radii*np.sin(angles)).flatten())
# Compute z to make the pringle surface.
z = np.sin(-x*y)

fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot_trisurf(x, y, z,color=(0.01,1,0.1), linewidth=0.2, antialiased=True)
plt.show()
```



❖ ***Plotting bar chart in 3D***

```
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure()
ax = fig.gca(projection='3d')
X = np.arange(-5, 5, 0.25)
Y = np.arange(-5, 5, 0.25)
X, Y = np.meshgrid(X, Y)
R = np.sqrt(X**2 + Y**2)
Z = np.sin(R)
colors = np.random.rand(40, 40, 4)
surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1, facecolors=colors,
linewidth=0, antialiased=False)
ax.set_zlim(-1.01, 1.01)
ax.zaxis.set_major_locator(LinearLocator(10))
ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))
plt.show()
```

❖ ***Animation***

To animate, we need to import the animation file as follows:
import matplotlib.animation as animation.
Then use the method **FuncAnimation().**
Definition:
FuncAnimation(fig, func, frames=None, init_func=None, fargs=None, save_count=None, **kwargs)

class **FuncAnimation(TimedAnimation):**

Makes an animation by repeatedly calling a function **func**, passing in (optional) arguments in **fargs**.

frames can be a generator, an iterable, or a number of frames.

init_func is a function used to draw a clear frame. If not given, the results of drawing from the first item in the frames sequence will be used. This function will be called once before the first frame.

If blit=True, **func** and **init_func** should return an iterable of drawables to clear.
****kwargs** include ‘**repeat**’, ‘**repeat_delay’**, and ‘**interval**’:

interval draws a new frame every **interval** milliseconds.
repeat controls whether the animation should repeat when the sequence of frames is completed.
repeat_delay optionally adds a delay in milliseconds before repeating the animation.

Example:

```
from mpl_toolkits.mplot3d import axes3d
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

def generate(X, Y, phi):
    R = 1 - np.sqrt(X**2 + Y**2)
    return np.cos(2 * np.pi * X + phi) * R

fig = plt.figure()
ax = axes3d.Axes3D(fig)

xs = np.linspace(-1, 1, 50)
ys = np.linspace(-1, 1, 50)
X, Y = np.meshgrid(xs, ys)
Z = generate(X, Y, 0.0)
wframe = ax.plot_wireframe(X, Y, Z, rstride=2, cstride=2)
ax.set_zlim(-1,1)

def update(i, ax, fig):
    ax.cla()
    phi = i * 360 / 2 / np.pi / 100
    Z = generate(X, Y, phi)
    wframe = ax.plot_wireframe(X, Y, Z, rstride=2, cstride=2)
    ax.set_zlim(-1,1)
    return wframe,

ani = animation.FuncAnimation(fig, update,frames=range(100),fargs=(ax, fig),
interval=100)
plt.show()
```

D) <u>Mayavi (using python2.7)</u>

Mayavi is an interactive 3D plotting and visualization package suited to handle large and complex data. See its document on http://docs.enthought.com/mayavi/mayavi/mlab.html

To use mayavi we import the following package:

import numpy as np
from mayavi import mlab

and below your code as follows:

mlab.clf() *, to clear current figure*
create plotting data
draw figure
mlab.show()

in Ipython we do the following:

%gui qt # indicate the use of ***qt***
import numpy as np
from mayavi import mlab
# and below your code

To have a control of the plotting screen use:

`mayavi.mlab.figure`*(figure=None, bgcolor=None, fgcolor=None, engine=None, size=(400,350))*

- **3D Points Plotting**

  mlab.points3d(x,y,z,color,scale_factor,line_width,mode='sphere',resolution=8)
  plots glyphs (like points) at the position of the supplied data.
  x,y,z: arrays of points coordinate as
  color:color (r,g,b) from 0 to 1, example (0,0,1) for blue
  scale_factor: point size as a float
  resolution: points smoothness , default=8 (integers)
  mode: primitive type to draw.default is 'sphere' but can be: '2darrow' , '2dcircle' , '2dcross'
  '2ddash' , '2ddiamond' , '2dhooked_arrow' ,'2dsquare' , '2dthick_arrow' , '2dthick_cross' '2dtriangle' , '2dvertex' , 'arrow' , 'axes' , 'cone' , 'cube' 'cylinder' ,'point' or 'sphere'.

  Example:

  ```
  import numpy as np
  from mayavi import mlab

  #clear the current figure
  mlab.clf()
  x=np.random.randint(10,size=10)
  y=np.random.randint(15,size=10)
  z=np.random.randint(20,size=10)
  mlab.points3d(x,y,z,color=(0,0,1),scale_factor=1.1,resolution=20)
  mlab.show()
  ```

❖ 3D Surface Plotting

mlab.surf(x,y,f,color,line_width,colormap,representation,warp_scale) *or* ***mlab.surf(s,…) , mlab.surf(x,y,s,…) , mlab.surf(x,y,f,…).   f=f(x,y)***
plots a surface using regularly-spaced elevation data supplied as a 2D array.
s is the elevation matrix, a 2D array, where indices along the first array axis represent x locations, and indices along the second array axis represent y locations.
Representation: figure as a 'surface', 'wireframe',or 'points'. Default='surface'
Warp_scale:scale of the z-axis

Example1: using numpy.mgrid()

```
import numpy as np
from mayavi import mlab

def f(x,y): return 0.5 +(x/3)**2 + (y/3)**2

# change the screen width=800, height=600
mlab.figure(size=(800,600))


#clear the current figure
mlab.clf()
x,y=np.mgrid[-5:5:0.5,-5:5:0.5]
#mlab.gcf() # generate the current frame
mlab.surf(x,y,f(x,y),representation='wireframe')
mlab.show()
```

Note: when using numpy.meshgrid() in mayavi, the data obtained for x,y,z,… must be transposed ;see example below

Example2: using numpy.meshgrid()

```
import numpy as np
from mayavi import mlab

def f(x,y): return 0.5 +(x/3)**2 + (y/3)**2

# change the screen width=800, height=600
mlab.figure(size=(800,600))
#clear the current figure
mlab.clf()
#x,y=np.mgrid[-5:5:0.5,-5:5:0.5]
x_grid=np.linspace(-5.,5.0,num=60)
y_grid=np.linspace(-5.,5.0,num=60)
x,y=np.meshgrid(x_grid,y_grid)
x=x.transpose() # transpose x for fit mayavi requirements
y=y.transpose()#transpose y for fit mayavi requirements
#mlab.gcf() # generate the current frame
mlab.surf(x,y,f(x,y), color=(0,0,1),representation='wireframe')
mlab.show()
```

- **3D lines , Curves(parametric)**

  Mlab.plot3d(x,y,z,s,color,color map,line width,representation,tube radius,tube sides)

  tube_radius: radius of the tubes used to represent the lines, If None, simple lines are used.

  tube_sides: number of sides of the tubes used to represent the lines; default: 6

  Example:

```
import numpy as np
from mayavi import mlab

# change the screen width=800, height=600
mlab.figure(size=(800,600))
#clear the current figure
mlab.clf()
PI=np.pi
t=np.linspace(0,1,num=1000)
n=30
x=t
y=4*PI*t + 4*PI*( np.cos(2*PI*n*t) - np.cos(2*PI*t))
z=np.cos(4*PI*t)+ 4*PI*(np.sin(2*PI*n*t) - np.sin(2*PI*t))
mlab.plot3d(x,y,z,color=(0.0,0.0,1.0),tube_radius=0.09)
mlab.show()
```

- **3D Mesh**

  mlab.mesh(x,y,z,color,colormap,mode,tube_radius,tube_sides,…..)

  plots a surface using grid-spaced data supplied as 2D arrays.

  Example: to draw

$$\begin{cases} x = 0.5\, sin^2(u)\, cos(2\, v) \\ y = 0.5\, sin^2(u)\, sin(2\, v) \\ z = sin(u)\, sin(v) \\ 0 \le u \le \pi \;,\; 0 \le v \le 2\pi \end{cases}$$

```
import numpy as np
from mayavi import mlab

# change the screen width=800, height=600
mlab.figure(size=(800,600))
#clear the current figure
mlab.clf()
cos = np.cos
sin = np.sin
pi=np.pi
u = np.linspace(0,pi, 50)
v = np.linspace(0,2*pi,50)
u,v=np.meshgrid(u,v)
x = 0.5*(sin(u))**2 *(cos(2*v))
y = 0.5*(sin(u))**2 *(sin(2*v))
z = sin(u)*sin(v)
x=x.transpose()
y=y.transpose()
z=z.transpose()
mlab.mesh(x,y,z)
mlab.show()
```

Example 2: to draw parametric surface

$$x = 6\cos(u)\cdot(1+\sin(u)) + r\cdot\cos(u)\cdot\cos(v) \qquad if \;\; 0 \le u < \pi$$
$$x = 6\cos(u)\cdot(1+\sin(u)) + r\cdot\cos(v+\pi) \qquad if \;\; \pi < u \le 2\pi$$
$$y = 16\cdot\sin(u) + r\cdot\sin(u)\cdot\cos(v) \qquad if \;\; 0 \le u < \pi$$
$$x = 16\cdot\sin(u) \;\; , \quad z = r\cdot\sin(v) \qquad if \;\; \pi < u \le 2\pi$$

$$r = 4 - a\cos(u)$$
$$a = 2$$

Example2:

```
import numpy as np
from mayavi import mlab

# change the screen width=800, height=600
mlab.figure(size=(800,600))
#clear the current figure
mlab.clf()

cos = np.cos
sin = np.sin
pi  =np.pi

u=np.linspace(0,2*pi,num=50)
v=np.linspace(0,2*pi,num=50)
u,v=np.meshgrid(u,v)
a=2
r=4-a*cos(u)



if u.all()>=0 and u.all()<pi :
      x = 6*cos(u)*(1+sin(u)) + r*cos(u)*cos(v)

if u.all()>pi and u.all()<=2*pi :
      x = 6*cos(u)*(1+sin(u)) + r*cos(v+pi)

if u.all()>=0 and u.all()<pi :
      y = 16*sin(u) + r*sin(u)*cos(v)

if u.all()>pi and u.all()<=2*pi :
      x = 16*sin(u)
     z=r*sin(v)

x=x.transpose()
y=y.transpose()
z=z.transpose()
mlab.mesh(x,y,z,representation='wireframe',color=(0,1,0))

mlab.show()
```

- 3D Contour Surface
  mlab.contour_surf(x,y,f,color,colormap, line_width,…) ,
  mlab.contour_surf(x,y,s,) or mlab.contour_surf(s,…) plots the contours of a surface using grid-spaced data for elevation supplied as a 2D array.

  Example: $f(x,y) = \sin^2(x) \cdot \cos(y)$

```
import numpy as np
from mayavi import mlab

# change the screen width=800, height=600
mlab.figure(size=(800,600))
#clear the current figure
mlab.clf()
cos = np.cos
sin = np.sin
x_grid = np.linspace(-3, 3,num=60)
y_grid = np.linspace(-3, 3,num=60)
x, y =np.meshgrid(x_grid, y_grid)
def f(x,y): return (sin(x)**2)*cos(y)
x=x.transpose()
y=y.transpose()
mlab.contour_surf(x,y,f)
mlab.surf(x,y,f,representation='wireframe')
mlab.show()
```

- **3D Flow**
  mlab.flow(u,v,w,…), mlab.flow(x,y,z,u,v,w,…) mlab.flow(x,y,x,f)
  creates a trajectory of particles following the flow of a vector field.
  Keywords:
  integration_direction: The direction of the integration. Must be ‘forward’ or ‘backward’ or ‘both’. Default: forward
  seedtype: the widget used as a seed for the streamlines. Must be ‘line’ or ‘plane’ or ‘point’ or ‘sphere’. Default: sphere .
  color,colormap,extent ……:

  Example: on the velocity field of a fluid flow $\vec{u}(u,v,w) = 0.5xy\vec{i} - y^2\vec{j} + yz^2\vec{k}$

```
import numpy as np
from mayavi import mlab
# change the screen width=800, height=600
mlab.figure(size=(800,600))
#clear the current figure
mlab.clf()
cos = np.cos
sin = np.sin
pi  =np.pi
x, y, z = np.mgrid[0:5:0.01, 0:5:0.1, 0:5:0.1]
u=0.5*x*y
v=-y**2
w=y*z**2
mlab.flow(u,v,w)
```

- **Quiver(vector field visualization)**

  mlab.quiver3d(u,v,w,…) , mlab.quiver3d(x,y,z,u,v,w,…), mlab.quiver3d(x,y,z,f,…)

  plots glyphs (like arrows) indicating the direction of the vectors at the positions supplied.
  u, v, w are numpy arrays giving the components of the vectors.If only 3 arrays, u, v, and w are passed, they must be 3D arrays, and the positions of the arrows are assumed to be the indices of the corresponding points in the (u, v, w) arrays.If 6 arrays, (x, y, z, u, v, w) are passed, the 3 first arrays give the position of the arrows, and the 3 last the components. They can be of any shape.If 4 positional arguments, (x, y, z, f) are passed, the last one must be a callable, f, that returns vectors components (u, v, w) given the positions (x, y, z).

  Key arguments:
  Color,colormap,line_width
  scale_factor: The scaling applied to the glyphs. the size of the glyph is by default calculated from the inter-glyph spacing Specify a float to give the maximum glyph size in drawing units
  scale_mode: the scaling mode for the glyphs (‘vector’, ‘scalar’, or ‘none’).

  Example:

```
import numpy as np
from mayavi import mlab

# change the screen width=800, height=600
mlab.figure(size=(800,600))
#clear the current figure
mlab.clf()
cos = np.cos
sin = np.sin
pi  =np.pi
x, y, z = np.mgrid[-2:3:1, -2:3:1, -2:3:1]
u=0.5*x*y
v=-y**2
w = np.zeros_like(z)
mlab.quiver3d(x, y, z, u, v, w, line_width=0.1,color=(0,1,0), scale_factor=0.5,
              scale_mode='scalar')

mlab.xlabel("x-axis")
mlab.ylabel("y-axis")
mlab.zlabel("z-axis")
mlab.orientation_axes()
mlab.show()
```

- **Mayavi figure decoration functions**

  Mayavi provides method for displaying text,title,axes,labels.
  xlabel(*text*, *object=None*) creates a set of axes if there isn’t already one, and sets the x label.
  ylabel(*text*, *object=None*) creates a set of axes if there isn’t already one, and sets the y label.
  zlabel(*text*, *object=None*) creates a set of axes if there isn’t already one, and sets the z label.
  colorbar(*object=None*, *title=None*, *orientation=None*, *nb_labels=None*, *nb_colors=None*, *label_fmt=None*) adds a colorbar for the color mapping of the given object.orientation:vertical|horizontal
  scalarbar(*object*=*None*, *title*=*None*, *orientation*=*None*, *nb_labels*=*None*, *nb_colors* =*None*, *label_*fmt=None*) adds a colorbar for the scalar color mapping of the given object.
  text(x,y,z,text,color,line_width,…) adds a text to a figure.

Text3d(x,y,z,text,color,line_width,orient_to_camera=’True’,…)
title(text,color,height,line_width,size,…) adds a title to a figure.
axes(color,extent,line_width,xlabel,ylabel,zlabel,…) creates axes for current object
Example: see example above on Quiver plot

- **Mayavi Animation**
  To create animation in Mayavi, use the method:
  mlab.animate(func=none,delay=500,ui=True)
  a decorator for animation (@mlab.animate())
  ui: display animation GUI if set ‘True’

  Below is a Mayavi animation framework:

```
from mayavi import mlab
mlab.clf()
  # initialization goes here (objects creation).
  # for instance, create body1 at (x1,y1,z1)
  mlab.animate()
  @mlab.animate(delay=100)
  def update(dt=duration_value):
  while True:
       # animation code goes below
        # next  position update  for each body defined goes below
        body1.mlab_source.reset(x=x1,y=y1,z=z1)

        yield
  update()
  mlab.show()
```

Example:

```
import numpy as np
from mayavi import mlab

cos = np.cos
sin = np.sin
mlab.clf()
#create body1 at (0,0,0)
x1=y1=z1=0.0
body1=mlab.points3d(x1,y1,z1,color=(0,0,1 ),scale_factor=0.3)
#create body1 at (2,0,4)
x2,y2,z2=2,0.0,4
body2= mlab.points3d(x2,y2,z2,color=(0,1,1 ),scale_factor=0.3)
mlab.animate()
@mlab.animate(delay=100)
def update(dt=0.025):

    t=0.0
    angle=0.0
    mlab.gcf()
    while True:
      t=t+dt
      angle +=dt
      global x1,y1,z1
```

```
        x1=cos(angle)+5*dt
        y1=sin(angle)+0.1*dt
        z1=z1+dt

        global x2,y2,z2
        x2=cos(10*angle)
        y2=sin(10*angle)
        z2=z2
        print(t)
        body1.mlab_source.reset(x=x1,y=y1,z=z1)
        body2.mlab_source.reset(x=x2,y=y2,z=z2)
        #f.scene.render()
        yield

    update()
    mlab.show()
```

E) PyQtGraph

To get the examples and tutorials ,open a python console and type:

```
import pyqtgraph.examples
pyqtgraph.examples.run()
```

some website example :

http://ruggero.sci.yokohama-cu.ac.jp/blog/pyQtGraph.php
http://nullege.com/codes/search/pyqtgraph.plot

F) Scipy

Scipy or Scientific Python is a python scientific computing package built on top of numpy.
It is scipy= numpy + more.It works very well with Simpy,Matplotlib,Pandas and various python scientific
Library. It has libraries dedicted for Linear Algebra,Calculus,Differential equations,Statistics,Computational
Geometry , Cryptography , Graph and Number theory and many more.
Here we will explore the Linear Algebra, differential and integral Calculus, and Statistics.

Scipy for Linear Algebra
We need to import the following packages for Linear algebra:
import numpy as np
from scipy import linalg as la

Matrix creation: to create a matrix we use numpy.matrix() , scipy.matrix() or numpy.mat()
scipy.matrix(data, dtype=none,copy=true) where data is a ndarray or a string between quotation marks. In the last case(string),the semi-colons means change in the rows, and the comas the change in the columns.

Example: $m = \begin{pmatrix} 1 & 0 & 5 \\ 0 & 3 & 4 \\ 1 & 2 & 3 \end{pmatrix}$ will be **m=numpy.matrix([[1,0,5] , [0,3,4] , [1,2,3]])**

or

m= numpy.matrix('1,0,5 ; 0,3,4 ; 1,2,3')

a column vector $\vec{v} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$ **will be v= numpy.matrix('1;2;3') or**

v= numpy.matrix([[1],[2],[3]])

a row vector $\vec{v} = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$ **will be v= numpy.matrix([[1],[2],[3]])**

or v= numpy.matrix('1,2,3')

- **Vectors operations:**
  Let consider the vectors $\vec{a} = (2,3)$ and $\vec{b} = (3,4)$
  PYTHONIC: a=numpy.matrix('2,3') b=numpy.matrix('3,4')
  Vector addition/subtraction : $\vec{a} + \vec{b} = (2,3) + (3,4) = (5,7)$
  PYTHONIC: a+b or a-b for the difference
  Inner product or vector dot product : $\vec{a} \bullet \vec{b} = (2,3) \bullet (3,4) = 6 + 12 = 18$
  PYTHONIC: numpy.inner(a,b) **not** a*b or numpy.dot(a,b)

- **Matrices operations:**
  Let the columns vectors $\vec{v} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$,$\vec{u} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$ and matrices $M = \begin{pmatrix} 1 & 2 \\ 0 & 3 \end{pmatrix}$,
  $N = \begin{pmatrix} 1 & 5 \\ 2 & -1 \end{pmatrix}$

PYTHONIC: v=np.matrix(' 2;3') , u =np.matrix('4;3') , M=mat('1,2;0,3') , N=mat('1,5 ;2,-1')

Scalar matrix multiplication: $2\vec{v} = 2 \cdot \begin{bmatrix} 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 4 \\ 6 \end{bmatrix}$; $2 \bullet M = 2 \bullet \begin{pmatrix} 1 & 2 \\ 0 & 3 \end{pmatrix} = \begin{pmatrix} 2 & 4 \\ 0 & 6 \end{pmatrix}$

PYTHONIC : 2*v , 2*M the * operator has been overloaded for this purpose.

Matrix addition/subtraction: $M + N = \begin{pmatrix} 1 & 2 \\ 0 & 3 \end{pmatrix} + \begin{pmatrix} 1 & 5 \\ 2 & -1 \end{pmatrix} = \begin{pmatrix} 2 & 7 \\ 2 & 2 \end{pmatrix}$

PYTHONIC: M+N , also the difference will be M-N

Matrix multiplication: $M \cdot N = \begin{pmatrix} 1 & 2 \\ 0 & 3 \end{pmatrix} \cdot \begin{pmatrix} 1 & 5 \\ 2 & -1 \end{pmatrix} = \begin{pmatrix} 5 & 3 \\ 6 & -3 \end{pmatrix}$

PYTHONIC: M*N or numpy.dot(M,N) or numpy.matmul(M,N) or the scipy version

Note: **numpy.multiply( )** performs components-wise multiplication that is

$$\begin{pmatrix} 1 & 2 \\ 0 & 3 \end{pmatrix}\begin{pmatrix} 1 & 5 \\ 2 & -1 \end{pmatrix} = \begin{pmatrix} 5 & 10 \\ 0 & -3 \end{pmatrix}$$

inner product(dot product) of columns vectors: $\vec{u}^t \cdot \vec{v} = \begin{bmatrix} 3 & 4 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \end{bmatrix} = 6 + 12 = 18$

PYTHONIC: u.T*v or u.transpose()*v or s=np.dot(u.T,v) , T is for transpose of matrix

Outer product of columns vectors: $\vec{u} \cdot \vec{v}^T = \begin{bmatrix} 3 \\ 4 \end{bmatrix} \begin{bmatrix} 2 & 3 \end{bmatrix} = \begin{pmatrix} 6 & 9 \\ 8 & 12 \end{pmatrix}$

PYTHONIC: u*v.T or numpy.dot(u,v.T)

Matrix-vector multiplication: $M \cdot \vec{v} = \begin{pmatrix} 1 & 2 \\ 0 & 3 \end{pmatrix} \begin{pmatrix} 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 8 \\ 9 \end{pmatrix}$

PYTHONIC: use numpy.matmul(M,v) or M*v

- **Matrices function**

Vectors norm: norm of $\vec{u} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$ is $\|\vec{u}\| = \sqrt{3^2 + 4^2} = 5$

PYTHONIC : use numpy.linalg.norm(u,ord=none,axis=None,keepdims=False) or
scipy.linalg.norm(). Ex : numpy.linalg.norm(u)

Matrix determinant: $\det(M) = \begin{vmatrix} 1 & 2 \\ 0 & 3 \end{vmatrix} = 3 - 0 = 3$

PYTHONC: use numpy.linalg.det(a,overwrite_a=False,check_finite=True) or scipy.linalg()
Ex: scipy.linalg.det(M)

Matrix inverse : inverse of $M = \begin{pmatrix} 1 & 2 \\ 0 & 3 \end{pmatrix}$ is $M^{-1} = \begin{pmatrix} 1 & -2/3 \\ 0 & 1/3 \end{pmatrix}$

PYTHONIC: use numpy.linalg.inv(a,overwrite_a=False,check_finite=True) or Scipy.linalg.inv()
Ex: scipy.linalg.inv(M) .

Matrix eigen numbers and eigen vectors :

Linalg.eig(M) computes the eigenvalues and right eigenvectors of asquare matrix M. See example from Numpy Section.

Scipy for Calculus

- **Polynomial definition and roots finding :**

We can use **numpy.poly1d(arr)** or **scipy.poly1d()** to create a polynomial, where arr is an array as: $arr = [a_n, a_{n-1}, a_{n-2}, ..., a_1,]$ that is $a_n \cdot x^n + a_{n-1} \cdot x^{n-1} + ... + a_2 \cdot x + a_1$

For instance arr=[2,3,4,5] will be $2x^3 + 3x^2 + 4x + 5$.

Polynomials can be added (+),multiplied(*),derived using arr.deriv() ,integrated using
Arr.integ()

```
Example:
import numpy as np
p=np.poly1d([2,3,4,5])
q=np.poly1d([1,-2,-9,18])
sum=p+q
prod=p*q
p_derivative=p.deriv()
p_integrate=p.integ()
sol=np.roots(q)

print("polynomial p is :\n" + "{}".format(p))
print("polynomial q is :\n" + "{}\n".format(q))
print("p+q=\n"+ "{} ".format(sum))
print("p*q=\n " + "{} ".format(prod))
print("derivative of p is :\n" + "{}".format(p_derivative))
print("derivative of p is :\n" + "{}".format(p_integrate))
print("roots of q are: {}".format(sol))

output:
    polynomial p is :
  3     2
2 x + 3 x + 4 x + 5
polynomial q is :
  3     2
1 x - 2 x - 9 x + 18
p+q=
  3     2
3 x + 1 x - 5 x + 23
p*q=
   6     5     4     3     2
2 x - 1 x - 20 x + 6 x + 8 x + 27 x + 90
derivative of p is :
  2
6 x + 6 x + 4
derivative of p is :
    4     3     2
0.5 x + 1 x + 2 x + 5 x
root of p are: [-3.  3.  2.]
```

❖ Derivative

scipy.misc.derivative(*func, x0, dx=1.0, n=1, args=(), order=3*)

Finds the n-th (n) derivative of a function at a point *x0*.

Given a function, use a central difference formula with spacing *dx* to compute the *n*-th derivative at *x0*.

x0: The point at which *n*-th derivative is found

dx : (float) spacing or stepsize,default is 1

n: integer , Order of the derivative. Default is 1

args : tuple, optional arguments

order : int, optional. Number of points to use; must be odd.

Example1:

```
 import numpy as np
from scipy.misc import derivative

  cos=np.cos
 def f(x):
    return x**3 + x*(1 + cos(x) )
 print(derivative(f, 1.0, n=5,order=7,dx=1))
```

❖ General purpose single integration:

To integrate just import the integrate library as follows:

from scipy import integrate

the scipy integrate has the following methods to integrate functions:

odeint	-- Integrate ordinary differential equations.
quad	-- General purpose integration.
dblquad	-- General purpose double integration.
tplquad	-- General purpose triple integration.
gauss_quad	-- Integrate func(x) using Gaussian quadrature of order n.
gauss_quadtol	-- Integrate with given tolerance using Gaussian quadrature.

Then choose any method of integrations: quadrature(i***ntegrate.quad),simpson(simp)***

For single integration use**:** ***value, error =integrate.*****quad(func, a, b[, args, full_output, ...])**

Example1: Compute $I = \int_1^2 (2x + x^3)dx$ **using *integrate.*quad()**

```
 import numpy as np
from scipy import integrate as integrate

def f(x): return 2*x+ x**3
val,err=integrate.quad(f,1,2)
print("integration value is {0} with error {1}".format(val,err)
output:  integration value is 6.75 with error 7.49400541622e-14
```

Example2*: compute* $I = \int_1^{+\infty} e^{-x^2} dx$ **using *integrate.*quad()**

```
import numpy as np
from scipy import integrate as integrate

inf=np.inf # infinity
e=np.e
def f(x): return e**-(x**2)
val,err=integrate.quad(f,1,inf)
print("integration value is {0} with error {1}".format(val,err))
output:  integration value is 0.13940279264 with error 3.07280182571e-10.
```

For double integration use : integrate.dblquad()
dblquad(func, a, b, gfun, hfun, args=(), epsabs=1.49e-08, epsrel=1.49e-08)
returns the double (definite) integral of func(x, y) from x = a to x=b and y =gfunc(x) to y=hfunc(x), that is $I = \int_{x=a}^{x=b} \int_{y=g(x)}^{y=h(x)} f(x,y)dxdy$.
Note that gfunc(x) and hfunc(x) which are respectively g(x) and h(x) are functions
to be defined using the lambda functions.

Example1: compute $I = \int_{x=0}^{x=1} \int_{y=1}^{y=2} (xy)\ dxdy$

```
import numpy as np
from scipy import integrate as integrate

def f(x,y): return x*y
val,err=integrate.dblquad(f,0,1,lambda x:1, lambda x:2)
print("integration value is {0} with error {1}".format(val,err))

output is :  integration value is 0.75 with error 8.32667268469e-1
```

Example2: compute $I = \int_{x=0}^{x=1} \int_{y=x}^{y=x^2} (xy)\ dxdy$

```
import numpy as np
from scipy import integrate as integrate
def f(x,y): return x*y
val,err=integrate.dblquad(f,0,1,lambda x:x, lambda x:x**2)
print("integration value is {0} with error {1}".format(val,err))

output :   integration value is -0.0416666666667 with error 4.62592926927e-
16
```

Example3 : Compute $I = \int_{0}^{\infty} \int_{1}^{\infty} \frac{e^{-xy}}{x^2}\ dxdy$

```
import numpy as np
from scipy import integrate as integrate

inf=np.inf # infinity
exp=np.exp
def f(x,y): return exp(-x*y)/x**2
val,err=integrate.dblquad(lambda x,y:f(x,y),0,inf,lambda y:1, lambda y:inf)
print("integration value is {0} with error {1}".format(val,err))

output:  integration value is 0.499999999909 with error 1.46408395125e-08
```

G) <u>SymPy</u>

SymPy is a Python symbolic mathematic library similar to Maples and Mathematica.
It can be used as a powerful calculator.
Unlike Maples and Mathematica , Sympy requires the symbols variable to be defined.

Symbols definition:
To define a variable symbol we import the ***Symbol*** library or use the following statement:
'*from sympy.abc import x,y*' to define 2 symbols x and y

Example1: using Symbol

```
from sympy import Integral, pprint,Symbol
x=Symbol('x')
y=Symbol('y')
pprint(x**2)
pprint(y**4-y**3 +3)
print("\n")
pprint(1/x)
print("\n")
pprint(Integral(x**2 + 1/x, x),use_unicode=True)
```

Example2: using ***sympy.abc import***

```
from sympy import Integral, pprint
from sympy.abc import x,y
pprint(x**2)
pprint(y**4-y**3 +3)
print("\n")
pprint(1/x)
print("\n")
pprint(Integral(x**2 + 1/x, x),use_unicode=True)
```

output :

x^2

$y^4 - y^3 + 3$

$\frac{1}{x}$

$\int \left(x^2 + \frac{1}{x} \right) dx$

Application to Algebra:

Here are all the import for the examples below:

import sympy as sy
from sympy import pprint,Symbol
x=Symbol('x')
y=Symbol('y')

- Expanding algebraic expression with ***sympy.expand(expr) on*** $(x-2y)^2$ ***:***

  ```
  m=(x-2*y)**2
  M=sy.expand(m)
  pprint(M)
  ```

  output: $x^2 - 4xy + 4y^4$

- Factoring algebraic expression with ***sympy.factor(expr) on*** $4xy^2 + 8y$***:***

```
expr=4*x*y**2+ 8*y
expr=sy.factor(expr,x,y)
pprint(expr)
output:4y(xy+2)
```

- Simplifying expression using ***sympy.simplify(expr) on*** $3x+y+4x-5y$

```
import sympy as sy

x,y=sy.symbols('x,y')#   Symbol('x')
expr=3*x + y + 4*x - 5*y
expr=sy.simplify(expr)
sy.pprint(expr)

output:  7x-4y
```

- Partial fraction decomposition using ***sympy.apart(expr) on*** $\frac{x}{(x-2)(x+3)}$***:***

```
import sympy as sy

x=sy.Symbol('x')
expr=x/(x-2)*(x+3)
expr=sy.apart(expr)
sy.pprint(expr)
```

ouput : $x+5+\frac{10}{x-2}$

- **Combine algebraic terms together using sympy.together(expr)on** $\frac{1}{x}+\frac{1}{y}+\frac{1}{z}$**:**

```
import sympy as sy

x,y,z=sy.symbols('x,y,z')
expr=1/x + 1/y+ 1/z
expr=sy.together(expr)
sy.pprint(expr)
```

output : $\frac{xy+xz+yz}{xyz}$

- Solving algebraic equation using ***sympy.solve(expr),***

```
import sympy as sy
x=sy.Symbol('x')
X=sy.solve(x**4 -10*x**2+9,x)
sy.pprint(X)
```

output : $x^4-10x^2+9=0$ ➔ x=1,-3,-1,3

Application to Calculus:

- Limit of a function using ***sympy.limit(function,variable,point)***:
-

  To find $\lim_{x->2}\frac{x+4}{x+1}=2 \quad and \quad \lim_{x->\infty}\frac{x+4}{x+1}=1$

```
import sympy as sy

x=sy.Symbol('x')
def f(x) : return (x+4)/(x+1)
# make f(x) symbolic
f=f(sy.Symbol('x'))
lim_at_2=sy.limit(f,x,2)
print("f(x)=\n")
sy.pprint(f)
print("\n" + "limit of f(x) at 2 is  {}".format(lim_at_2))
oo=sy.oo  # infinity
lim_at_oo=sy.limit(f,x,oo)
print("limit of f(x) at infinity is  { }".format(lim_at_oo))
```

- Differentiate using ***sympy.diff(function,variable,order)***

  Example on $f(x)=x\cos(x)$

```
import sympy as sy

x=sy.Symbol('x')
cos=sy.cos

def f(x) : return x*cos(x)

# make f(x) symbolic
f=f(sy.Symbol('x'))
first_derivative=sy.diff(f,x)
print("f(x)=\n")
sy.pprint(f)
print("\n" + "the first derivative of f is :")
sy.pprint(first_derivative)
second_derivative=sy.diff(f,x,2)
print("\n" + "the second derivative of f is :")
sy.pprint(second_derivative)
```

- Integration using **sympy.integral(func,(x,lower,upper)** and sympy.integrate(func,(x,lower,upper))

```
import sympy as sy

x=sy.Symbol('x')

def f(x) : return 3*x**2 + 9
# make f(x) symbolic
f=f(sy.Symbol('x'))
F=sy.Integral(f,(x,1,2))
print("f(x)=\n")
sy.pprint(f)
print("\n" + "the integral of  f(x) from 1 to 2 is :")
sy.pprint(F)
print("\n=")
value=sy.integrate(f,(x,1,2))
sy.pprint(value)
```

output : $\int_1^2 \left(3x^2 + 9\right) dx = 16$

- Series expansion using ***sympy.serie(func,x,point,order)***

```
import sympy as sy

x=sy.symbols('x')
f=1/(x-1)
s=sy.series(f,x,0,4)
print("f(x)=\n")
sy.pprint(f)
print("\n 4th order series expansion of f(x)  around 0 \n")
sy.pprint(s)
s=sy.series(f,x,2,4)
print("\n 4th order series expansion of f(x)  around 2 \n")
sy.pprint(s)
```

output:

$$f(x) = \frac{1}{x-1}$$

$$-1 - x - x^2 - x^3 + O(x^4)$$

$$3 + (x-2)^2 - (x-2)^3 - x + O((x-2)^4)$$

Application to Linear Algebra:

Most Linear Algebra matrix and vectors operation in Sympy are the same in Numpy and Scipy. Here we will only show how to create a matrix and vector and some of their algebra

Example:

```
using Sympy by example.
import sympy as sy

# create matrix A  using Sympy.Matrix()
A=sy.Matrix([ [1,2],[3,1] ])
print("Matrix A=")
sy.pprint(A)
# create matrix B  using Sympy.Matrix()
print("\n"+"Matrix B=")
B=sy.Matrix([ [4,0],[1,2] ])
sy.pprint(B)
print("\n"+"A*B=")
# multiply A and B using *
C=A*B
sy.pprint(C)
print("\n"+"3A - 2B=")
S=3*A-2*B
sy.pprint(S)
#compute determinant of A using Sympy.det()
print("\n"+"determinant of A=")
sy.pprint(A.det())
#compute inverse of A using Sympy.inv()
inv_A=A.inv()
print("\n"+"inverse of A=")
sy.pprint(inv_A)
```

output :

$$Matrix\ \ A=\begin{pmatrix}1 & 2\\ 3 & 1\end{pmatrix}$$

$$Matrix\ \ A \bullet B=\begin{pmatrix}6 & 4\\ 13 & 2\end{pmatrix}$$

$$3\cdot A-2\cdot B=\begin{pmatrix}-5 & 6\\ 7 & -1\end{pmatrix}$$

determinant of A=-5

$$\textit{inverse of A:}\begin{pmatrix}-1/5 & 2/5\\ 3/5 & -1/5\end{pmatrix}$$

H) Numba , Python optimization

Numba gives you the power to speed up your applications with high performance functions written directly in Python. With a few annotations, array-oriented and math-heavy Python code can be just-in-time compiled to native machine instructions, similar in performance to C, C++ and Fortran, without having to switch languages or Python interpreters.

Numba works by generating optimized machine code using the LLVM compiler infrastructure at import time, runtime, or statically (using the included pycc tool). Numba supports compilation of Python to run on either CPU or GPU hardware, and is designed to integrate with the Python scientific software stack.

We need first to import numba by doing this:
from numba import jit
by using the decorator ***@jit ,*** a function is marked by Numba for optimization.We will see how it is used in different code optimization options.

- Basic usage:
  - Lazy compilation
    In this case the **@jit** decorator let Numba decide on when and how to optimize your code.
    Example1

```
from numba import jit
import numpy as np

@jit
def prod(x,y):
     return x[0]*y[0] + x[1]*y[1] + x[2]*y[2] + x[3]*y[3]

a=np.array([2,4,5,1])
b=np.array([-3,2,1,2])
c=prod(a,b)
print(c)

output : 9
```

  - Eager compilation
    In this case you are in change. You decide on the function signature as in this example.
    Example2:

```
from numba import jit,float32
import numpy as np

@jit(float32(float32,float32))
def prod(x,y):
     return x[0]*y[0] + x[1]*y[1] + x[2]*y[2] + x[3]*y[3]

 a=np.array([2.0,4.0,5.0,1.0]) # all float
 b=np.array([-3.0,2.0,1.0,2.0]) # all float,if int will result into error
 c=prod(a,b)
 print(c)
```

- **Calling and inlining other functions**

  Functions compiled by numba can call other functions. These called functions need to be jit-TED to optimally run.

  Example:

```
from numba import jit
import numpy as np

@jit
def ratio(x,y):
   ret=0
   if x<=y and y!=0 : ret=x/y
   return  ret

@jit
def compute_angle(x,y):
   val=ratio(x,y)    # ratio()  has been jit-TED before using it here for faster run .
   return np.math.degrees(np.math.acos(val))

x =input(" enter the first number: ")
y =input(" enter the first number: ")
value=compute_angle(x,y)
```

  Output:

```
enter the first number: 3
enter the first number: 9
angle value is 90.0
```

- **Functions signature specifications**

  explicit **@jit** signatures can use a number of types. Here are some common ones:

  - **void** : functions returning nothing.
  - **intp** and **uintp** are pointer-sized integers (signed and unsigned, respectively)
  - **intc** and **uintc** are equivalent to C int and unsigned int integer types
  - **int8, uint8, int16, uint16, int32, uint32, int64, uint64** are fixed-width integers of the corresponding bit width (signed and unsigned)
  - **float32** and **float64** are single- and double-precision floating-point numbers, respectively
  - **complex64** and **complex128** are single- and double-precision complex numbers, respectively
  - array types can be specified by indexing any numeric type, e.g. **float32[:]** for a one-dimensional single-precision array or **int8[:,:]** for a two-dimensional array of 8-bit integers.

❖ Compilation options

Some keyword arguments can be passed to @jit to define a compilation options.
They are : **nopython , nogil and cache.**
nopython mode and object mode define the two compilation modes in Numba**.**
object mode : it is a slow compilation mode that generate python objects and uses the python C API to perform operations on the python objects.
nopython mode : it generates faster code, but has some limitation that can force Numba to fall back to object mode. To prevent the fall back use **@jit(nopython=True)**
Example:

```
from numba import jit,float64
import numpy as np

@jit(float64[:,:](float64[:,:] ,float64[:,:] ),nopython=True)
def mult(A,B):
   return np.dot(A,B)

M=np.mat('1.0,2.0;2.0,4.0')
N=np.mat('1.0,0.0;2.0,5.0')
Q=mult(M,N)
print("{}\n".format(M))
print("{}\n".format(N))
print(Q)
```

output :

```
[[ 1.  2.]
 [ 2.  4.]]

[[ 1.  0.]
 [ 2.  5.]]

[[  5.  10.]
 [ 10.  20.]]
```

nogil option: the global interpreter lock (**GIL**) is a mechanism to synchronize the execution of thread so that only one native thread can execute at a time at the expense of much of the parallelism found in multi-core processors computers.
Setting ***@jit(nogil=True)*** will help Numba to take advantage of the multi-core systems.
Example:

```
 from numba import jit ,int32

@jit( nogil=True)
  def mult(x,y):
       return x*y

@jit( int32[:]( int32[:],int32[:] ), nogil=True)
  def add(x,y):
       return x+y
```

cache option: set **@jit(cache=True)** to avoid compilation everytime a python program is invoked.
Example:

```
@jit( cache=True)
  def mult(x,y):
       return x*y
```

- Creating Numpy universal function(ufunc)

Using **vectorize()** you write your function as operating over input scalars, rather than arrays. Numba will generate the surrounding loop allowing efficient iteration over the actual inputs. If using eager compilation, then square brackets []should be used instead of parenthesis () as follows: @vectorize([float64(float64,float64)]) not @vectorize((float64(float64,float64))) because we are manipulating array instead of individual scalar

Example1: with lazy compilation

```
from numba import jit,vectorize
import numpy as np

@vectorize
def f(x): return x**2-3*x + 1

v=np.mat("1,2,0,4")
w=f(v)
print(w)
```

Example2: with eager compilation

```
from numba ,vectorize,int32
import numpy as np

@vectorize( [int32(int32) ])
def f(x): return x**2-3*x + 1

v=np.mat("1,2,0,4")
w=f(v)
print(w)
```

- **Compiling Python class with jit@class**

Numba can mark a class for optimization with type specification for each field using the decorator **jit@class( ).**The specification field (spec) is provided as a list of 2-tuples. The tuples contain the name of the field and the numba type of the field as follows:

```
spec=[ ( 'value', float64 ),  → scalar field of type float64
        ('value', int32),       → scalar field of type int32
        ('array', float32[:] ),  → array field of type float32
         ('array',float64[:,:] )    → nd-array field of  type float64
      ]
jit@class(spec)
class yourClass(object):
     __init__(self,float64Field,int32Field,arrayfloat32Field, arrayfloat64Field):
     ……
```

The **jitclass** is imported as
follows: ***from numba import jitclass***

Example:

```
  from numba import jitclass  # import the decorator
  from numba import float32   # import the types
  import numpy as np

  #class field specifications
  spec=[('x',float32),
       ('y',float32),
       ('z',float32),
       ('arr',float32[:]),
       ]

  @jitclass(spec)
  class vector3d(object):
       #@void(float_,float_,float_)
       def __init__(self,x=0.0,y=0.0,z=0.0):
          self.x=x
          self.y=y
          self.z=z
          self.arr=np.array([self.x,self.y,self.z],dtype=np.float32)

       def add(self,v):
           return vector3d(self.x +v.x , self.y + v.y , self.z + v.z)

       def createMatrix(self):
          return self.arr

  def main():
     v1=vector3d(1.0,0.0,3.0)
     v2=vector3d(1.0,4.0,2.0)
     w=v1.add(v2)
     print("v1 =( {0} , {1} , {2} )".format(v1.x,v1.y,v1.z))
     print("v2 =( {0} , {1} , {2} )".format(v2.x,v2.y,v2.z))
     print("v1+v2 =( {0} , {1} , {2} )".format(w.x,w.y,w.z))
     M=v1.createMatrix()
     print("v1 in matrix form is:")
     print(M)

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

output :

```
v1 =( 1.0 , 0.0 , 3.0 )
v2 =( 1.0 , 4.0 , 2.0 )
v1+v2 =( 2.0 , 4.0 , 5.0 )
v1 in matrix form is:
[ 1.  0.  3.]
```

I) Mypy, Python(3.5+) optional static type-checking

Mypy is an optional static type checking mechanism for Python 3.5 and above.
It helps in adding type hints to Python programs to make it more readable, and to check for errors in your code. Mypy does not interfere with python code, but acts like a comment around the codes . below are the most buit-in types in Mypy

Type	Description
Int	integer of arbitrary size
Float	floating point number
Bool	boolean value
Str	unicode string
Bytes	8-bit string
Object	an arbitrary object (object is the common base class)
List[str]	list of str objects
Dict[str, int]	dictionary from str keys to int values
Iterable[int]	iterable object containing ints
Sequence[bool]	sequence of booleans
Any	dynamically typed value with an arbitrary type

First we need to import the type hinting as follows:
from typing Import Iterable ,Dict, List, sequences # to name a few
or just:
import typing

- ***Variable type annotation:***

```
# annotation is similar to arguments to functions
name: str = "Eric Idle"

# class instances can be annotated as follows
mc : MyClass = MyClass()

# tuple packing can be done as follows
tu: Tuple[str, ...] = ('a', 'b', 'c')

# annotations are not checked at runtime
year: int = '1972'  # error in type checking, but works at runtime

# these are all equivalent
hour = 24 # type: int
hour: int; hour = 24
hour: int = 24

# you do not (!) need to initialize a variable to annotate it
a: int # ok for type checking and runtime

# which is useful in conditional branches
child: bool
if age < 18:
   child = True
else:
   child = False
```

```
# annotations for classes are for instance variables (those created in __init__ or
__new__)
class Battery:
   charge_percent: int = 100  # this is an instance variable with a default value
   capacity: int  # an instance variable without a default

# you can use the ClassVar annotation to make the variable a class variable instead of
an instance variable.
class Car:
   seats: ClassVar[int] = 4
   passengers: ClassVar[List[str]]

# You can also declare the type of an attribute in __init__
class Box:
   def __init__(self) -> None:
      self.items: List[str] = []
```

- **Example: function takes 2 floats and returns a float**

  ```
  def add(x:float,y:float)->float:
  return x+y
  ```

- **Example: function takes a string and returns a string**

  ```
  display(name:str)->str:
  return " your name is" + name
  ```

- **Example : function with iterable**

  ```
  from typing import Iterable

  def doit(x:float)->Iterable[float]:
     return 6*[x]
  c=10
  d=doit(c)
  print(d)  # output  is  [10, 10, 10, 10, 10, 10]
  ```

- @typing.**overload**
  The `@overload` decorator allows describing functions and methods that support multiple different combinations of argument types. A series of `@overload`-decorated definitions must be followed by exactly one non-`@overload`-decorated definition (for the same function/method).
  See example below

Example:

```
from typing import overload
import numpy as np

# the overloaded functions come first before the un-overloaded function
@overload
def add(x:str,y:str)->str:
   return x+y

@overload
def add(x:[int],y:[int])->[int]:
   return x+y


def add(x:int,y:int)->int:
   return x+y

def main():
   a=4
   b=6
   c=add(a,b)
   print(c)
   str1="Aaron"
   str2="Bahin"
   str=add(str1,str2)
   print(str)
   arr1=np.array([1,2,3],dtype=int)
   arr2=np.array([4,0,1],dtype=int)
   arr=add(arr1,arr2)
   print(arr)

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

- Type aliases
  A type alias is defined by assigning the type to the alias. In this example, Vector and List[float] will be treated as interchangeable synonyms.

  Example:

```
from typing import List,Iterable
import numpy as np

#create an alias type list of float
Mylist=List[float]
vector=Iterable[float]

def add_list(L1:Mylist ,L2:Mylist)->Mylist:
   return L1+ L2

def add_vector(L1:vector ,L2:vector)->vector:
   return L1+ L2
```

```
def main():

   v1=np.array([1,2,3],dtype=float)#.tolist()
   v2=np.array([4,0,1],dtype=float)#.tolist()
   v=add_vector(v1,v2)
   print("vector addition is : {}".format(v))

   arr1=np.array([1,2,3],dtype=float).tolist()
   arr2=np.array([4,0,1],dtype=float).tolist()
   arr=add_vector(arr1,arr2)
   print("list addition is: {}".format(arr))

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

Output:

```
 vector addition is : [ 5.  2.  4.]
list addition is: [1.0, 2.0, 3.0, 4.0, 0.0, 1.0]
```

- Creating new type using **NewType()**
  A new type can be created using ***NewType()***

  Example:

```
from typing import NewType
from array import array

#create a new type array of integer
Array_int=NewType("Array_int",[int])

def mult(a:int,arr:Array_int)->Array_int:
   return a*arr

def main():
   c=2
   arr=array('i',[1,2,4])
   d=mult(c,arr)
   print(d)


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

output: array('i', [1, 2, 4, 1, 2, 4])

- User defined-generic type

  A user-defined class can be defined as a generic class using TypeVar( )

  Example:

```
from typing import TypeVar
vector3=TypeVar("vector3")

class vector3(object):
   # constructor with default value zero (float)
   def __init__(self,x:float(0.0),y:float(0.0),z:float(0.0)):
      self.x=x
      self.y=y
      self.z=z

   def add(self,v:vector3)->vector3:
      return vector3(self.x + v.x,self.y + v.y ,self.z + v.z)

   def __str__(self)->str:
      return "( {0} , {1} , {2} )".format(self.x,self.y,self.z)

   @staticmethod
   def dot(v1:vector3,v2:vector3)->float:
    return v1.x*v2.x+v1.y*v2.y+v1.z*v2.z

def main():
   v1=vector3(1.0,2.0,3.0)
   v2=vector3(2.0,1.0,-2.0)
   d=vector3.dot(v1,v2)
   sum=v1.add(v2)
   print("v1*v2= {}".format(d) )
   print("v1+v2=",end="")
   print(sum)

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

  Output:

```
v1*v2= -2.0
v1+v2=( 3.0 , 3.0 , 1.0 )
```

J) **Pandas**
K) **Others**
L) **Scikits**
M)