```
import numpy as np
a = np.array([2,3,4,5])
print(a)
print(type(a))
print(a.ndim)
print(a.shape)
```

```
[2 3 4 5]
<class 'numpy.ndarray'>
1
(4,)
```

```
a = np.array([2+1j,3,4,5],dtype = complex)
print(a)
```

```
[2.+1.j 3.+0.j 4.+0.j 5.+0.j]
```

```
import numpy as np
datetime1=np.array(['2023-03-03T12:00:00','2023-06-
07T03:01:00'],dtype="datetime64")
a=np.diff(datetime1)
print(datetime1.dtype)
print(a.dtype)
```

```
datetime64[s]
timedelta64[s]
```

```
from numpy import random
s = random.rand()#1st number is between 0 and 7,.......
b=random.randint([2,3,7,9])
print(s)
print(b)
```

```
0.6441086278883184
[1 2 1 5]
```

```
a = np.zeros((3,4)) #row,column
print(a)
```

```
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

```
a = np.ones((3,2,4)) #groups,rows,columns
print(a)
```

```
[[[1. 1. 1. 1.]
  [1. 1. 1. 1.]]

 [[1. 1. 1. 1.]
```

```
  [1. 1. 1. 1.]]

 [[1. 1. 1. 1.]
  [1. 1. 1. 1.]]]
```

```
a = np.arange(20,50,5) #generate numbers betwen 20 and 50 but in 5's
print(a)
```

```
[20 25 30 35 40 45]
```

```
#numbers divisible by n:
a = np.arange(5,1000,5)
print(a)
```

```
[  5  10  15  20  25  30  35  40  45  50  55  60  65  70  75  80  85
90
  95 100 105 110 115 120 125 130 135 140 145 150 155 160 165 170 175
180
 185 190 195 200 205 210 215 220 225 230 235 240 245 250 255 260 265
270
 275 280 285 290 295 300 305 310 315 320 325 330 335 340 345 350 355
360
 365 370 375 380 385 390 395 400 405 410 415 420 425 430 435 440 445
450
 455 460 465 470 475 480 485 490 495 500 505 510 515 520 525 530 535
540
 545 550 555 560 565 570 575 580 585 590 595 600 605 610 615 620 625
630
 635 640 645 650 655 660 665 670 675 680 685 690 695 700 705 710 715
720
 725 730 735 740 745 750 755 760 765 770 775 780 785 790 795 800 805
810
 815 820 825 830 835 840 845 850 855 860 865 870 875 880 885 890 895
900
 905 910 915 920 925 930 935 940 945 950 955 960 965 970 975 980 985
990
 995]
```

```
a = np.linspace(0.11,2.2,8) #start,stop,n
print(a)
```

```
[0.11       0.40857143 0.70714286 1.00571429 1.30428571 1.60285714
 1.90142857 2.2       ]
```

```
a = np.arange(6) # 1d array
print(a)
print('\n')

b = np.arange(12).reshape(4,3) # 2d array
print(b)
print('\n')
```

```
c = np.arange(24).reshape(2,3,4) # 3d array
print(c)
```

```
[0 1 2 3 4 5]


[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]


[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]

 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
```

```
a = np.arange(6) # 1d array
print(a)
print(a[4])
```

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

```
b = np.arange(12).reshape(4,3) # 2d array
print(b)
print(b[1][1])
```

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

```
c = np.arange(24).reshape(2,3,4) # 3d array
print(c)
print(c[0][1][3])
```

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

 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
7
```

Operations

```
a = np.array( [20,30,40,50] )
b = np.array( [4,3,2,1] )
c = (a-b) #also try +,*,/,**,exp,add,sqrt np.fn()
print(c)
```

```
[16 27 38 49]
```

```
a = [400,4,25]
print(np.sqrt(a))
```

```
[20.  2.  5.]
```

```
#product-element*,matrix@,.dot
A = np.array( [[1,1],
               [0,1]] )
B = np.array( [[2,0],
               [3,4]] )
print(A*B)
print(A@B)
c = A.dot(B)
print(c)
```

```
[[2 0]
 [0 4]]
[[5 4]
 [3 4]]
[[5 4]
 [3 4]]
```

```
#min,max,axis,sum,cumsum
#axis 0 = col, axis 1 = row, axis 2 = dimension
from numpy import random
a = np.ones((2,3))
print(a)
print(a.sum())
print(a.sum(axis=1))
print(a.sum(axis=0))
```

```
[[1. 1. 1.]
 [1. 1. 1.]]
6.0
[3. 3.]
[2. 2. 2.]
```

```
a = np.floor(10*np.random.random((2,12)))
print(a)
```

```
[[5. 1. 9. 6. 2. 2. 1. 6. 7. 4. 0. 7.]
 [1. 5. 5. 8. 7. 1. 2. 5. 4. 2. 0. 7.]]
```

b = np.array[25,289,361,81] #find square roots and iterate through result values output:5 square is 25....

```
a = np.array([[1,5,8],[2,1,5],
              [2,3,6],[4,3,8]])
print(a)
print(a.max())
print(a.max(axis=0)) #column wise
print(a.max(axis=1)) #row wise
```

```
[[1 5 8]
 [2 1 5]
 [2 3 6]
 [4 3 8]]
8
[4 5 8]
[8 5 6 8]
```

```
a = np.array([[1,5,8],[2,1,5],
              [2,3,6],[4,3,8]])
print(a)
print(a.min())
print(a.min(axis=0)) #column wise
print(a.min(axis=1)) #row wise
```

```
[[1 5 8]
 [2 1 5]
 [2 3 6]
 [4 3 8]]
1
[1 1 5]
[1 1 2 3]
```

```
a = np.array([[1,5,8],[2,1,5],
              [2,3,6],[4,3,8]])
print(a)
print('\n')
print(a.cumsum(axis=0))
print('\n')
print(a.cumsum(axis=1))
```

```
[[1 5 8]
 [2 1 5]
 [2 3 6]
 [4 3 8]]
```

```
[[ 1  5  8]
 [ 3  6 13]
 [ 5  9 19]
 [ 9 12 27]]


[[ 1  6 14]
 [ 2  3  8]
 [ 2  5 11]
 [ 4  7 15]]
```

```
#cumulative sum
a = np.array([2,3,5,6])
print(a.cumsum())
```

```
[ 2  5 10 16]
```

Shape and size manipulation

```
a = np.array([34,35,36,37,38,39])

a = a.reshape(3,2)
print(a)

b = a.T
print(b)
print(b.shape)
```

```
[[34 35]
 [36 37]
 [38 39]]
[[34 36 38]
 [35 37 39]]
(2, 3)
```

```
a = np.array([34,35,36,37,38,39])
a.reshape(3,2) #need to be saved with another or same variable
print(a)
print(a.shape)
```

```
[34 35 36 37 38 39]
(6,)
```

```
a = np.array([34,35,36,37,38,39])
a.resize(3,2) #modifies the original array
print(a)
print(a.shape)
```

```
[[34 35]
 [36 37]
 [38 39]]
(3, 2)
```

array_joins

```
a = np.array([34,35,36,37,38,39])
a.resize(2,3)
b = np.array([4,5,6,7,8,9])
b.resize(2,3)

print(np.vstack((a,b))) #columns
print("\n")
print(np.hstack((a,b))) #rows
```

```
[[34 35 36]
 [37 38 39]
 [ 4  5  6]
 [ 7  8  9]]


[[34 35 36  4  5  6]
 [37 38 39  7  8  9]]
```

```
a = np.arange(30).reshape(2,3,5)
print(a)
print("\n")
print(np.dstack(a))
#number of rows becomes number of groups
#columns become rows
#group becomes columns
```

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

 [[15 16 17 18 19]
  [20 21 22 23 24]
  [25 26 27 28 29]]]


[[[ 0 15]
  [ 1 16]
  [ 2 17]
  [ 3 18]
  [ 4 19]]
```

```
 [[ 5 20]
  [ 6 21]
  [ 7 22]
  [ 8 23]
  [ 9 24]]

 [[10 25]
  [11 26]
  [12 27]
  [13 28]
  [14 29]]]
```

```
a = np.floor(10*np.random.rand(8,4))
print(a)
#print(np.vsplit(a,4)) #vsplit(arrayname,pieces)
print(np.vsplit(a,(3,5))) #vsplit(arrayname,(row1,row2......)
```

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

```
a = np.floor(10*np.random.random((2,12)))
print(a)

print("\n")
print(np.hsplit(a,3)) # Split a into 3

print("\n")
print(np.hsplit(a,(3,7))) # Split a after the third and the seventh
column
```

```
[[2. 0. 5. 4. 7. 3. 5. 9. 6. 5. 5. 8.]
 [2. 7. 8. 7. 7. 5. 3. 2. 5. 7. 1. 5.]]
```

```
[array([[2., 0., 5., 4.],
       [2., 7., 8., 7.]]), array([[7., 3., 5., 9.],
       [7., 5., 3., 2.]]), array([[6., 5., 5., 8.],
```

```
       [5., 7., 1., 5.]])]


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

Indexing

```
a = np.arange(12)**2 # the first 12 square numbers
print(a)
i = np.array( [ 1,1,3,8,5 ] ) # an array of indices
print(a[i])

j = np.array( [ [ 3,4,2], [ 9,7,11 ] ] ) # a bidimensional array of
indices
print(a[j])

k = np.array([[[3,4,2],[9,7,11],[0,10,1]],[[1,2,3],[7,8,9],[9,0,10]]])
print(a[k])
```

```
[  0   1   4   9  16  25  36  49  64  81 100 121]
[ 1  1  9 64 25]
[[  9  16   4]
 [ 81  49 121]]
[[[  9  16   4]
  [ 81  49 121]
  [  0 100   1]]

 [[  1   4   9]
  [ 49  64  81]
  [ 81   0 100]]]
```

```
a = np.arange(12).reshape(3,4)
print(a)

i = np.array( [ [0,1], # indices for the first dim of a
                [1,2] ] )
j = np.array( [ [2,1], # indices for the second dim
                [3,3] ] )

print(a[i,j]) # i and j must have equal shape
```

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

```
data = np.sin(np.arange(20)).reshape(5,4)
print(data)
print(data.argmax(axis=0))
```

```
[[ 0.          0.84147098  0.90929743  0.14112001]
 [-0.7568025  -0.95892427 -0.2794155   0.6569866 ]
 [ 0.98935825  0.41211849 -0.54402111 -0.99999021]
 [-0.53657292  0.42016704  0.99060736  0.65028784]
 [-0.28790332 -0.96139749 -0.75098725  0.14987721]]
[2 0 3 1]
```

```
#inverse matrix
a = np.array([[1,2],[3, 4]])
print(a)
print(np.linalg.inv(a))
```

```
[[1 2]
 [3 4]]
[[-2.   1. ]
 [ 1.5 -0.5]]
```

Numpy Joining arrays

```
import numpy as np
a1 = np.array([[1, 2], [3, 4]])
a2 = np.array([[5, 6], [7, 8]])
a3 = np.concatenate((a1, a2), axis=1)
print(a3)
```

```
[[1 2 5 6]
 [3 4 7 8]]
```

```
#search
arr = np.array([1, 2, 3, 4, 5, 4, 4])
x = np.where(arr == 4) #outputs index where item is 4
print(x)
```

```
(array([3, 5, 6], dtype=int64),)
```

Find the indexes where the values are even:

```
a = np.array([34,56,7,17,88,91])
#output=[0,1,4]
print(np.where(a%2 == 0))
```

```
(array([0, 1, 4], dtype=int64),)
```

```
#used only for already sorted list
a = np.array([6, 7, 8, 9, 10])
```

```
x = np.searchsorted(a, 10)
print(x)
```

```
4
```

```
#sorting array
import numpy as np
arr = np.array(['banana', 'cherry', 'apple'])
print(np.sort(arr))

arr = np.array([True, False, True])
print(np.sort(arr))
```

```
['apple' 'banana' 'cherry']
[False  True  True]
```

```
#2d sort - sorting happens within row
arr = np.array([[3, 2, 4], [5, 0, 1]])
print(np.sort(arr))
```

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

```
#array filter
arr = np.array([40, 42, 50, 44, 67, 78])
x = [True, False, True, False,True, True] #filter list
newarr = arr[x]
print(newarr)
```

```
[40 50 67 78]
```

```
a1 = np.array([1, 2, 3, 4, 5, 9, 6, 7])
# Create an empty list
filter_1 = []

# go through each element in array
for element in a1:
# if the element is completely divisble by 2, set the value to True,
otherwise False
    if element % 2 == 0:
        filter_1.append(True)
    else:
        filter_1.append(False)

n = a1[filter_1]

print(filter_1)
print(n)
```

```
[False, True, False, True, False, False, True, False]
[2 4 6]
```

Random

```
from numpy import random
x = random.choice([3, 5, 7, 9])
print(x)
```

```
7
```

```
y = random.rand() #float 0 to 1
print(y)
```

```
0.3807942990293054
```

```
z = random.randint(50) #range is 0 to 50
print(z)
```

```
44
```

```
a = random.randint(50, size = (5))
print(a)
```

```
[17 23  5 44  4]
```

```
b = random.randint(50, size = (3,2))
print(b)
```

```
[[ 5 45]
 [10 18]
 [30  4]]
```

```
from numpy import random
x = random.choice([3, 5], p=[0.2, 0.8], size=(3,5)) #p=probability
# 0.8 chances for 5 i.e, in total 15 elements 0.8*15 elements are 5
print(x)
```

```
[[5 5 5 5 5]
 [3 5 5 5 5]
 [5 5 5 3 5]]
```

```
a1 = np.array([1, 2, 3, 4, 5])
random.shuffle(a1)
print(a1)
```

```
[5 3 2 1 4]
```

Distributions

```
from numpy import random
x = random.normal(size=(2, 3)) #Gaussian distribution
print(x)
```

```
[[ 1.26652435  0.29618208  0.46349014]
 [-0.29835636  1.75036867  0.18275066]]
```

```
x = random.normal(loc=1, scale=2, size=(2, 3)) #loc=mean, scale =
stddeviation
print(x)
#generating 6 gaussian  values which can have mean=1 and std=2
```

```
[[3.20444992 4.15648084 3.26710656]
 [4.55776321 1.63582992 1.64425546]]
```

```
x = random.binomial(n=12, p=0.5, size=10) #n=number of trails,
p=probability,
print(x)
```

```
[1 5 4 8 3 6 5 5 7 7]
```

```
x = random.poisson(lam=2, size=10)
print(x)
```

```
[1 1 2 2 2 1 1 1 2 2]
```

Universal functions - ufuncs are used to implement vectorization in NumPy which is way faster than iterating over elements.

```
x = np.array([1, 2, 3, 4])
y = np.array([4, 5, 6, 7])
z = []
for i, j in zip(x, y):
    z.append(i + j)
print(z)
```

```
[5, 7, 9, 11]
```

```
x = [1, 2, 3, 4]
y = [4, 5, 6, 7]
z = np.add(x, y)
print(z)
```

```
[ 5  7  9 11]
```

```
#use user-defined functions
def myadd(x, y):
    return x+y

my = np.frompyfunc(myadd, 2, 1) #func_name,inputs,outputs
print(my([1, 2, 3, 4], [5, 6, 7, 8]))
```

```
[6 8 10 12]
```

```
x = [1, 2, 5, 8]
y = [4, 5, 6, 7]
z = np.subtract(y, x)
print(z)
```

```
[ 3  3  1 -1]
```

```
a1 = np.array([10, 20, 30, 40, 50, 60])
a2 = np.array([20, 21, 22, 23, 24, 25])
n = np.multiply(a1, a2)
print(n)
```

```
[ 200  420  660  920 1200 1500]
```

```
a1 = np.array([10, 21, 30, 40, 50, 60])
a2 = np.array([20, 21, 2, 20, 25, 25])
n = np.divide(a1, a2)
print(n)
```

```
[ 0.5  1.  15.   2.   2.   2.4]
```

```
a1 = np.array([10, 21, 30, 40, 50, 60])
a2 = np.array([20, 21, 2, 20, 25, 25])
n = np.mod(a1, a2)
print(n)
```

```
[10  0  0  0  0 10]
```

```
a1 = np.array([10, 21, 30, 40, 50, 60])
a2 = np.array([3,  2,  2,  7,  10, 7])
n = np.divmod(a1, a2)
print(n)
```

```
(array([ 3, 10, 15,  5,  5,  8]), array([1, 1, 0, 5, 0, 4]))
```

```
a1 = np.array([1,-1,2,-8.3,-3,-4]) #float and numeric |a|
a2 =np.absolute(a1)
print(a2)
```

```
[1.  1.  2.  8.3 3.  4. ]
```

```
#rounding
a1 = np.trunc([-3.1666, 3.6667])
print(a1)
```

```
[-3.  3.]
```

```
#rounding
a1 = np.fix([-3.1666, 3.6667])
print(a1)
```

```
[-3.  3.]
```

```
#rounding
a1 = np.around(3.16636, 4)
print(a1)
```

```
3.1664
```

```
#rounding
a1 = np.floor([-3.1666, 3.6667])
print(a1)
```

```
[-4.  3.]
```

```
#rounding
a1 = np.ceil([-3.1666, 3.1])
print(a1)
```

```
[-3.  4.]
```

```
#Logarithms
a1 = np.arange(1, 10)
print(a1)
print(np.log2(a1)) #base2
```

```
[1 2 3 4 5 6 7 8 9]
[0.         1.         1.5849625  2.         2.32192809 2.5849625
 2.80735492 3.         3.169925  ]
```

```
#Logarithms
a1 = np.arange(1, 10)
print(a1)
print(np.log10(a1)) #base10
```

```
[1 2 3 4 5 6 7 8 9]
[0.         0.30103    0.47712125 0.60205999 0.69897    0.77815125
 0.84509804 0.90308999 0.95424251]
```

```
#Logarithms-base e
a1 = np.arange(1, 10)
print(a1)
print(np.log(a1)) #base e
```

```
[1 2 3 4 5 6 7 8 9]
[0.         0.69314718 1.09861229 1.38629436 1.60943791 1.79175947
 1.94591015 2.07944154 2.19722458]
```

```
#products
a1 = np.array([1, 2, 3, 4])
a2 = np.array([5, 6, 7, 8])
x = np.prod([a1, a2])
print(x)
```

```
40320
```

```
#products
a1 = np.array([1, 2, 3, 4])
a2 = np.array([5, 6, 7, 10])
x = np.prod([a1, a2],axis=1)
print(x)
```

```
[  24 2100]
```

```
a2 = np.array([5, 6, 7, 10])
x = np.cumprod(a2)
print(x)
```

```
[   5   30  210 2100]
```

```
a1 = np.array([10, 15, 25, 15]) #e2-e1
newarr = np.diff(a1)
print(newarr)
```

```
[  5  10 -10]
```

```
num1 = 455
num2 = 665
x = np.lcm(num1, num2)
print(x)
```

```
8645
```

```
num1 = 455
num2 = 665
x = np.gcd(num1, num2)
print(x)
```

```
35
```

```
#applying gcd and lcm on arrays
#reduce=> multiple inputs one output
a1=np.array([12,24,6])
x = np.lcm.reduce(a1)
print(x)
```

```
24
```

```
a1=np.array([12,24,6])
x = np.gcd.reduce(a1)
print(x)
```

```
6
```

```
#trigonometry
x = np.sin(0)
print(x)
```

```
0.0
```

```
x = np.sin(np.pi/2)   #90 degrees
print(x)
```

```
1.0
```

```
a = np.array([np.pi/2, np.pi/3, np.pi/4, np.pi/5]) #sin 90,60,45,36
x = np.sin(a)
print(x)
```

```
[1.         0.8660254  0.70710678 0.58778525]
```

```
a = np.array([90, 180, 270, 360])
x = np.deg2rad(a)
print(x)
```

```
[1.57079633 3.14159265 4.71238898 6.28318531]
```

```
a = np.array([np.pi/2, np.pi/3, np.pi/4, np.pi/5])
x = np.rad2deg(a)
print(x)
```

```
[90. 60. 45. 36.]
```

Find hypotenues if base and perpendicular side are given

```
a = 8
b=6
hypotnus = np.hypot(a,b)
print(hypotnus)
```

```
10.0
```

```
#set
a = np.array([1, 1, 1, 2, 3, 4, 5, 5, 6, 7])
```

```
x = np.unique(a)
print(x)
```

```
[1 2 3 4 5 6 7]
```

```
a1 = np.array([[1, 2, 3, 4],[3,9,10,11]])
a2 = np.array([[3, 7, 5, 6],[5,3,8,10]])
x = np.union1d(a1, a2)
print(x)
```

```
[ 1  2  3  4  5  6  7  8  9 10 11]
```

```
a1 = np.array([1, 2, 3, 4])
a2 = np.array([3, 4, 5, 6])
x = np.intersect1d(a1, a2)
print(x)
```

```
[3 4]
```

```
a1 = np.array([1, 2, 3, 4])
a2 = np.array([3, 4, 5, 6])
x = np.setdiff1d(a2, a1)
print(x)
```

```
[5 6]
```

To find only the values that are NOT present in BOTH sets:

a = [1,2,3,4] b = [3,4,6,7] output [1,2,6,7]

```
a = np.array([1,2,3,4])
b = np.array([3,4,6,7])
print(np.setxor1d(a, b, assume_unique=True))
#symmetric difference
```

```
[1 2 6 7]
```

```
#ex1
# Importing the NumPy library
import numpy as np
l1 = np.log(1e-50)
l2 = np.log(2.5e-50)

print(np.logaddexp(l1, l2))
print(np.logaddexp2(l1, l2))
```

```
-113.87649168120691
-113.59955522772194
```

```
#ex2
# Importing the NumPy library
import numpy as np
print(np.true_divide((np.arange(10)), 3))
```

```
[0.         0.33333333 0.66666667 1.         1.33333333 1.66666667
 2.         2.33333333 2.66666667 3.        ]
```

```
#ex3
import numpy as np
x = np.array([-.7, -1.5, -1.7, 0.3, 1.5, 1.8, 2.0])
print(np.rint(x))
```

```
[-1. -2. -2.  0.  2.  2.  2.]
```

```
#ex4
# Importing the NumPy library
import numpy as np

# Creating NumPy arrays 'x' and 'y'
x = np.array([1, 4, 0], float)
y = np.array([2, 2, 1], float)

print(np.inner(x, y))

print(np.outer(x, y))

print(np.cross(x, y))
```

```
10.0
[[2. 2. 1.]
 [8. 8. 4.]
 [0. 0. 0.]]
[ 4. -1. -6.]
```

More!

```
lis = [1,2,3,4]
a = np.array(lis,dtype=np.int16)
print(a)
print(type(a))
print(a.dtype)
```

```
[1 2 3 4]
<class 'numpy.ndarray'>
int16
```

```
a1 = np.full((2,2), 7)
print(a1)
print(a1.itemsize) #bytes
```

```
[[7 7]
 [7 7]]
4
```

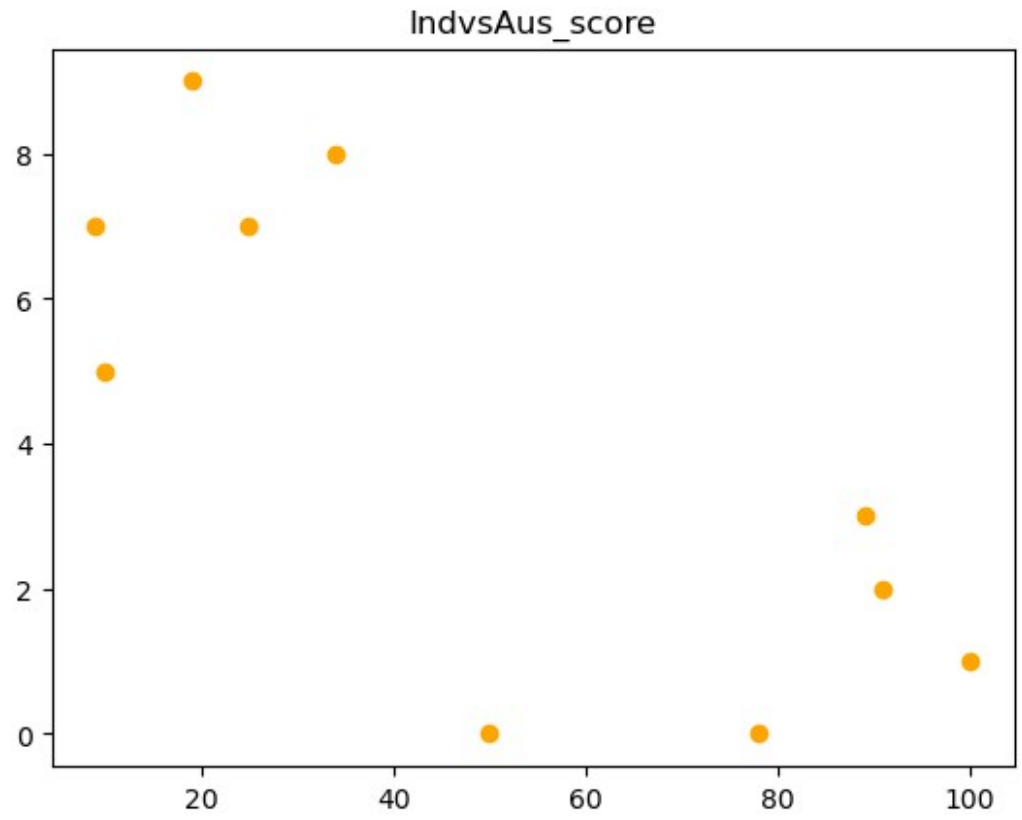
```
a1 = np.eye(4)
print(a1)
```

```
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
```

```
x = [1,2,3]
a = np.asarray(x)
print (a)
print(type(a))
```

```
[1 2 3]
<class 'numpy.ndarray'>
```

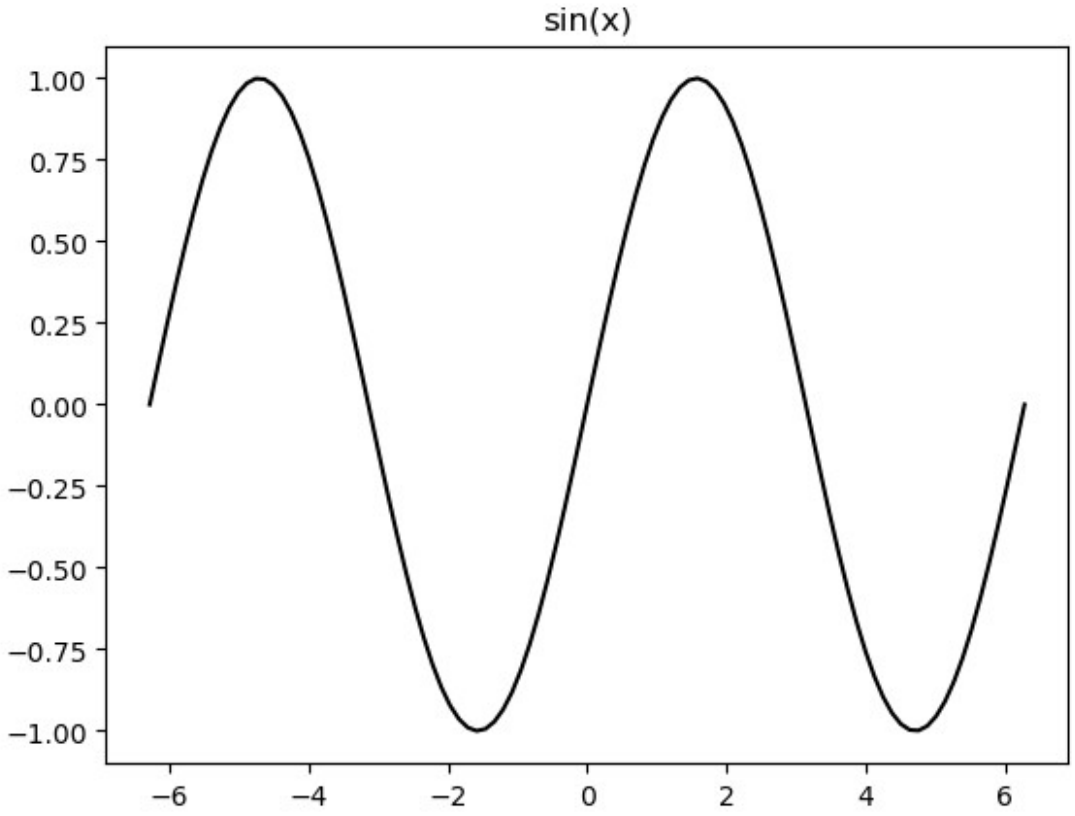
```
#cricket
import numpy as np
import matplotlib.pyplot as plt
runs = np.array([100,50,91,78,89,25,34,19,9,10])
w = np.array([1,0,2,0,3,7,8,9,7,5])
plt.scatter(runs,w,color='orange')
plt.title('IndvsAus_score')
plt.show()
```



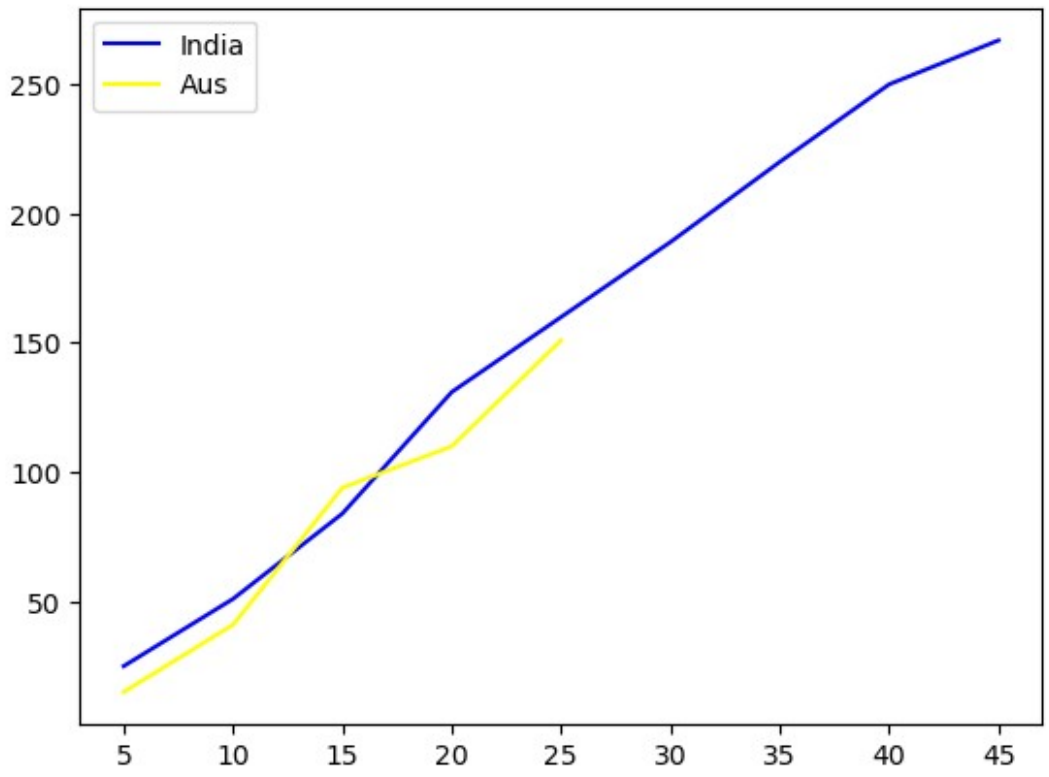

```
import numpy as np
import matplotlib.pyplot as plt
# Generate array of 200 values between -pi & pi
tigar = np.linspace(-2*np.pi, 2*np.pi, 100)
print(tigar)

plt.plot(tigar, np.sin(tigar) ,color='black') # SIN
plt.plot(x,y,,color,labels....)
plt.title("sin(x)")
# Display plot
plt.show()
```

```
[-6.28318531 -6.15625227 -6.02931923 -5.9023862  -5.77545316 -
5.64852012
 -5.52158709 -5.39465405 -5.26772102 -5.14078798 -5.01385494 -
4.88692191
 -4.75998887 -4.63305583 -4.5061228  -4.37918976 -4.25225672 -
4.12532369
 -3.99839065 -3.87145761 -3.74452458 -3.61759154 -3.4906585  -
3.36372547
 -3.23679243 -3.10985939 -2.98292636 -2.85599332 -2.72906028 -
2.60212725
 -2.47519421 -2.34826118 -2.22132814 -2.0943951  -1.96746207 -
1.84052903
```

```
 -1.71359599 -1.58666296 -1.45972992 -1.33279688 -1.20586385 -
1.07893081
 -0.95199777 -0.82506474 -0.6981317  -0.57119866 -0.44426563 -
0.31733259
 -0.19039955 -0.06346652  0.06346652  0.19039955  0.31733259
0.44426563
  0.57119866  0.6981317   0.82506474  0.95199777  1.07893081
1.20586385
  1.33279688  1.45972992  1.58666296  1.71359599  1.84052903
1.96746207
  2.0943951   2.22132814  2.34826118  2.47519421  2.60212725
2.72906028
  2.85599332  2.98292636  3.10985939  3.23679243  3.36372547
3.4906585
  3.61759154  3.74452458  3.87145761  3.99839065  4.12532369
4.25225672
  4.37918976  4.5061228   4.63305583  4.75998887  4.88692191
5.01385494
  5.14078798  5.26772102  5.39465405  5.52158709  5.64852012
5.77545316
  5.9023862   6.02931923  6.15625227  6.28318531]
```



```
#plotting all in one graph
# use : plt.legend(loc='best')
#plt.plot(x,y,color,label)
import numpy as np
import matplotlib.pyplot as plt
#creating x
overs = np.arange(5,50,5)
overs_a = np.arange(5,30,5)
#creating y
runs_i = np.array([25,51,84,131,160,189,220,250,267])
runs_a = np.array([15,41,94,110,151])
wickets = np.array([12,32,96])
#plotting
plt.plot(overs,runs_i,color='blue',label='India')
plt.plot(overs_a,runs_a,color='yellow',label = 'Aus')
#combining two graphs
plt.legend(loc='best')
#displaying the final graph
plt.show()
```




```
sal = np.array([12000,60000,80000,76000])
wd = np.array([210,300,230,256])
years = [1,2]
bonus=[]
for i,j in zip(sal,wd):
```

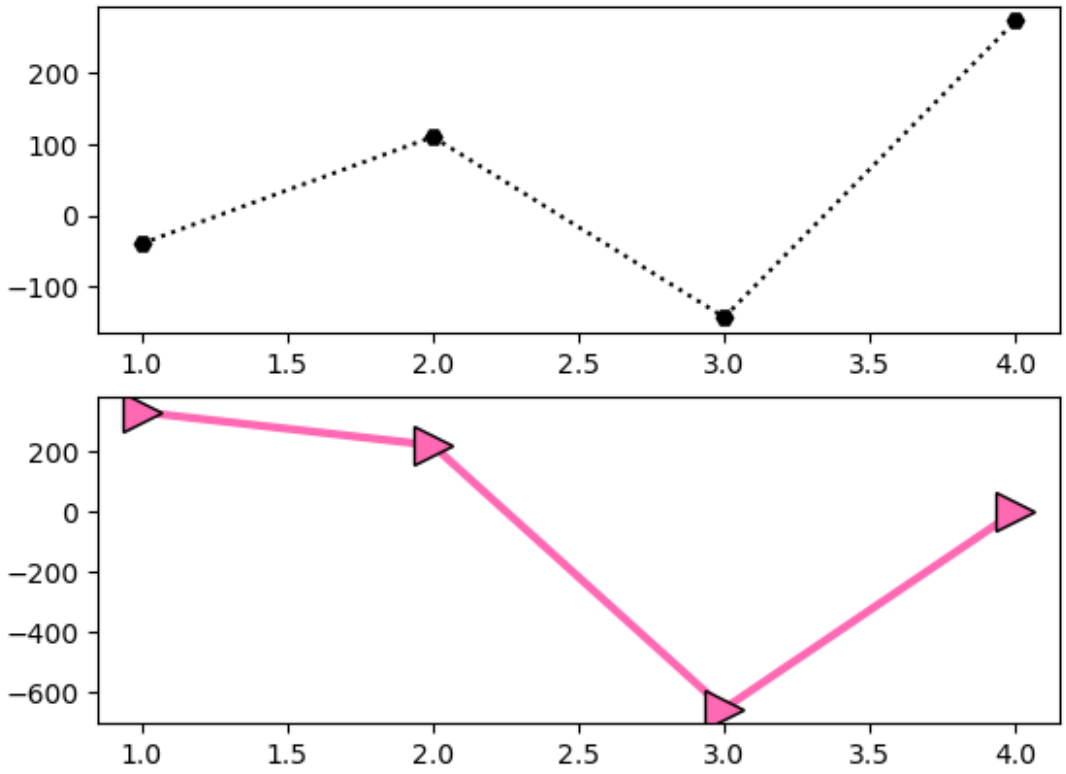
```
    if j > 250:
        i = i+i*0.1
        bonus.append(i)
    else:
        i = i-i*0.1
        bonus.append(i)
print(bonus)
```

```
[10800.0, 66000.0, 72000.0, 83600.0]
```

create a line graph showing increase or decrease in profit between two companies in lakhs over 10 years. a = [230,560,780,127,128...]

a_profits = [] year = [1,2,3,4,5,6,7,8,9,10]

```
import matplotlib.pyplot as plt

a = [230,560,780,127,128]
b = [200,160,270,127,400]
years = [1,2,3,4]

profit_a = [(a[i]-a[i-1]) for i in range(1,len(a))]
profit_b = [(b[i]-b[i-1]) for i in range(1,len(b))]
plt.subplot(2,1,2)
plt.plot(years,profit_a,color='hotpink',linewidth = '3',label =
'CompanyA',marker='>',ms='15',mec='k')
plt.subplot(2,1,1)
plt.plot(years,profit_b,color='black',linestyle='dotted',label =
'CompanyB',marker = 'H')


plt.show()
```



marker: marker ='*', ms=10 (marker size), mec= 'black'(marker color) mfc ='blue' (marker fill color)

linestyle linestyle = 'dotted'

linewidth: linewidth = '20'

Grid: plt.grid()

combining two plots 1 graph = legend() 2 graphs = subplot(a,b,c) before plot() function a = rows,b= columns, c= position

```
#inverse trig
print(np.arcsin(1)) #sin^-1
print(np.arccos(1))
print(np.arctan(0))
```

```
1.5707963267948966
0.0
0.0
```

Important Statistics Functions in NumPy

numpy.mean() : This function is used to calculate the mean of data.

numpy.average() : To calculate the average of any data.

numpy.median() : To calculate the median.

numpy.mode() : To calculate the mode of any data.

numpy.var() : To calculate the variance of any data.

numpy.std() : To calculate the standard deviation.

numpy.percentile() : To calculate percentile values.

numpy.corrcoef() : To calculate the correlation coefficient.

```
a = np.array([23,56,78,12,45,89,10,89,34,89,23,77])
#mean
print(np.mean(a))
#median
print(np.median(a))
#mode
#print(np.mode(a))
```

```
52.083333333333336
50.5
```

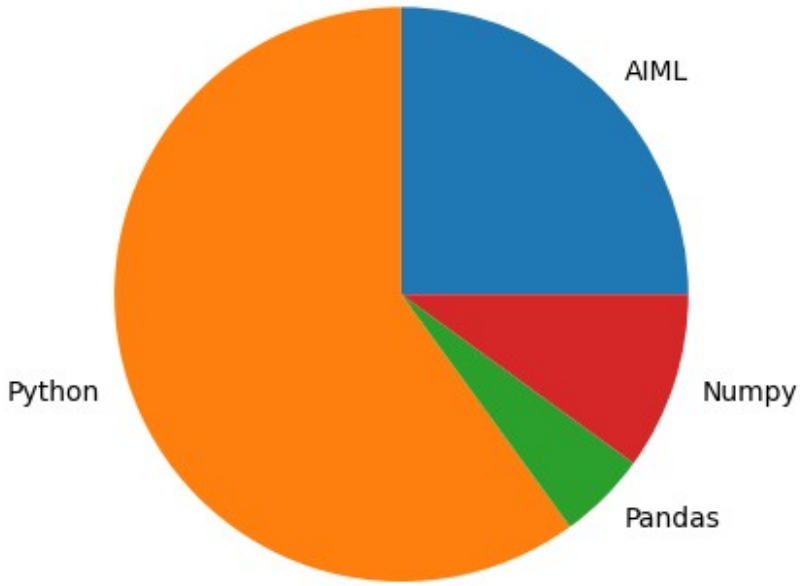
```
a = np.array([23,56,78,12,45,89,10,89,34,89,23,77])
b = np.array([23,56,78,12,40,90,10,85,31,89,23,77])
#var
print(np.var(a))
#std
print(np.std(a))
#corrcoef
print(np.corrcoef(a,b))
```

```
903.5763888888888
30.059547383300515
[[1.        0.9981131]
 [0.9981131 1.       ]]
```

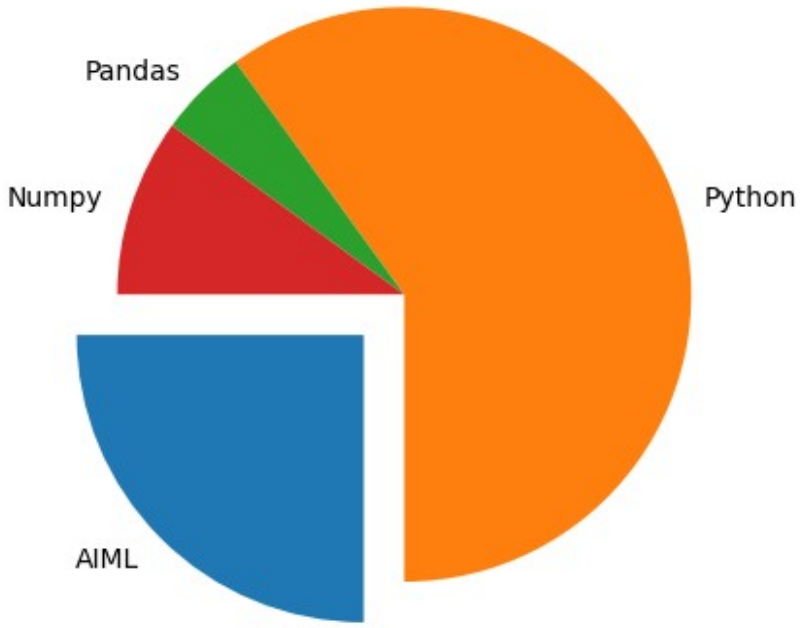
```
#vector length : |V| = √(x2 + y2 + z2)
a = np.array([2,5,7])
print(sqrt((sum(a**2))))
```

```
8.831760866327848
```

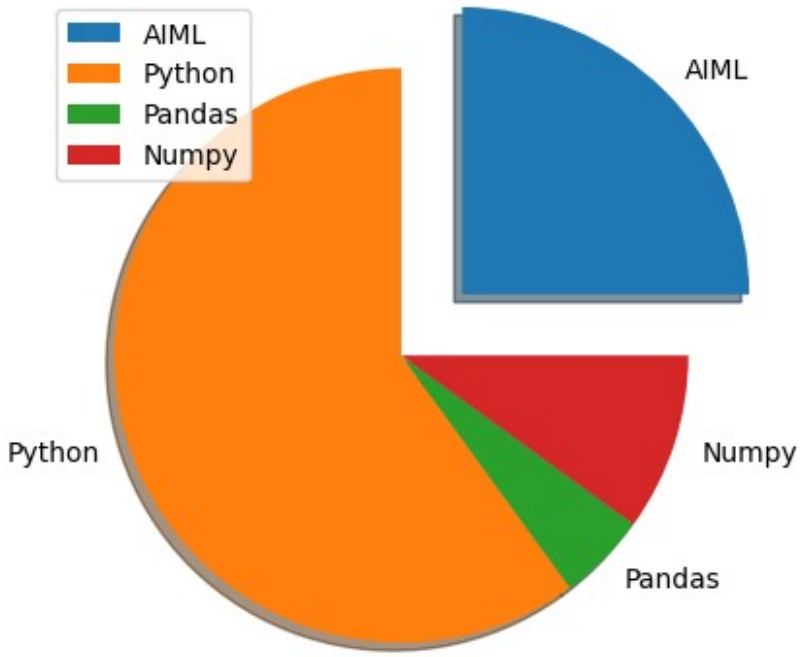
```
a = np.array([25,60,5,10])
labe = ["AIML","Python","Pandas","Numpy"]
plt.pie(a,labels = labe)
plt.show()
```



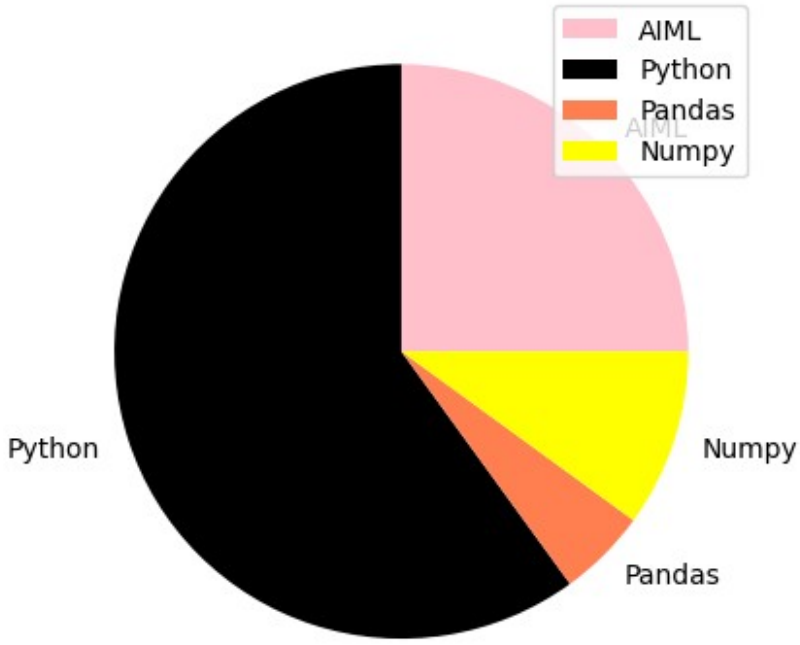

```
#explode
a = np.array([25,60,5,10])
labe = ["AIML","Python","Pandas","Numpy"]
explo = [0.2,0,0,0]
plt.pie(a,labels = labe,explode = explo,startangle = 180)
plt.show()
```




```
a = np.array([25,60,5,10])
labe = ["AIML","Python","Pandas","Numpy"]
explo = [0.3,0,0,0]
plt.pie(a,labels = labe,explode = explo,shadow=True)
plt.legend()
plt.show()
```




```
a = np.array([25,60,5,10])
labe = ["AIML","Python","Pandas","Numpy"]
color = ['pink','black','coral','yellow']
plt.pie(a,labels = labe,colors= color)
plt.legend()
plt.show()
```



Exercises

1. replace all the odd numbers in a numpy array with -1 ip = [0,1,2,3,4,5,6,7,8,9] op = [0,-1,2,-1,4,-1,6,-1,8,-1]
2. Create a numpy array with complex datatype and extract only real part
3. Create a 3D array and extract the unique elements
4. Create an array of elements and extract the numbers > 10 with their indices
5. Sort an array along columns {2D}
6. Plotting sales of shampoo for 12 months with labels, legend on right bottom, dotted and a pentagon marker
7. degreed to radians convertor

```
a = np.array([2+1j])
print(a.real)
```

```
[2.]
```

```
a=np.array([180,90,180])
print(np.deg2rad(a))
```

```
[3.14159265 1.57079633 3.14159265]
```

```
from numpy import *
a=array([0,10,20,20,30,40])
b=where(a>10)
print(b)
print(a[b])
```

```
(array([2, 3, 4, 5], dtype=int64),)
[20 20 30 40]
```

```
a=ceil(10*random.random(10))
print(a)
a[a%2!=0]=-1
print(a)
```

```
[8. 6. 9. 9. 9. 7. 6. 8. 4. 6.]
[ 8.  6. -1. -1. -1. -1.  6.  8.  4.  6.]
```

```
a=ceil(10*random.random(24)).reshape(2,3,4)
print(a)
print(unique(a))
```

```
[[[ 3.  6.  4.  7.]
  [ 1.  2.  9.  6.]
  [ 2.  1.  2.  4.]]

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