NumPy

1)what is Numpy?

- Numpy is the core library for scientific and numeric computing in python. it provides high performance multidimentional array object and tool for working with arrays. Its main object is the multi-dimention array.
- It is a table of elements(usually numbers),all of the same type, indexed by a tuple of positive integers. in Numpy dimensions are called axes.

2)Why should we use Numpy array when we have python list?

- Fast
- convenient
- Less Memory

```
import numpy as np

a = np.array([1,2,3])
a
```

```
array([1, 2, 3])
```

```
a[0]
```

```
1
```

```
import time
import sys

#to check the size diff in list and array
b=range(1000)
print(sys.getsizeof(5)*len(b))
c= np.arange(1000)
print(c.size*c.itemsize)
```

```
28000
4000
```

```
size = 100000
l1= range(size)
l2= range(size)
A1 = np.arange(size)
A2= np.arange(size)

#to check the time required by list and array
start = time.time()
result =[(x+y) for x,y in zip(l1,l2)]
print("python list took:",(time.time()-start)*1000)
start=time.time()
```

```
result = A1+A2
print("numpy array took:",(time.time()-start)*1000)
```

```
python list took: 7.002115249633789
numpy array took: 1.0173320770263672
```

```
 a= np.array([[1,2],[3,4],[5,6]])
a.ndim  # for checking dimention of the array
```

```
2
```

```
a.itemsize  # for item size
```

```
4
```

```
a.shape  #shape
```

```
(3, 2)
```

```
a= np.array([[1,2],[3,4],[5,6]], dtype=np.float64) #changing the data type
a
```

```
array([[1., 2.],
       [3., 4.],
       [5., 6.]])
```

```
a= np.array([[1,2],[3,4],[5,6]] , dtype=np.complex128)
a
```

```
array([[1.+0.j, 2.+0.j],
       [3.+0.j, 4.+0.j],
       [5.+0.j, 6.+0.j]])
```

```
np.zeros((3,4))
```

```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

```
np.ones((3,4), dtype=np.int_)
```

```
array([[1, 1, 1, 1],
       [1, 1, 1, 1],
       [1, 1, 1, 1]])
```

```
l= range(5)
l
```

```
range(0, 5)
```

```
np.arange(5)
```

```
array([0, 1, 2, 3, 4])
```

```
 print('concatination example:')
print(np.char.add(['hello','hi'],['abc','xyz']))
```

```
concatination example:
['helloabc' 'hixyz']
```

```
print(np.char.multiply('Hello ',4))
```

```
Hello Hello Hello Hello
```

```
print(np.char.center("hello",20,fillchar="-"))
```

```
-------hello--------
```

```
 print(np.char.title("hello how are you doing"))
 print(np.char.lower("hello how are you doing"))
 print(np.char.upper("hello how are you doing"))
 print(np.char.split("hello how are you doing"))
 print(np.char.strip(["hello", "how", "hare", "hyou", "hdoing"],'h'))
print(np.char.join([':'],['dmydj']))
print(np.char.replace('He is a very good dancer','is','was'))
```

```
Hello How Are You Doing
hello how are you doing
HELLO HOW ARE YOU DOING
['hello', 'how', 'are', 'you', 'doing']
['ello' 'ow' 'are' 'you' 'doing']
['d:m:y:d:j']
He was a very good dancer
```

Array Manipulation

changing shape

```
a=np.arange(9)
print("The original array:", a)
b= a.reshape(3,3)
print("reshape:\n", b)
print(b.flatten()) #for flattening it
print(b.flatten(order='F')) #flattening in y axis
```

```
The original array: [0 1 2 3 4 5 6 7 8]
reshape:
 [[0 1 2]
 [3 4 5]
 [6 7 8]]
[0 1 2 3 4 5 6 7 8]
[0 3 6 1 4 7 2 5 8]
```

```
a = np.arange(12).reshape(4,3)
print(a)
print('---------transpose---------\n',np.transpose(a))
# a = np.arange(12).reshape(4,2) it will give error
b=np.arange(8).reshape(4,2)
print('------reshaping-----\n')
c= b.reshape(2,2,2)
c
```

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

```
array([[[0, 1],
        [2, 3]],

       [[4, 5],
        [6, 7]]])
```

```
np.rollaxis(c,2,1)
```

```
array([[[0, 2],
        [1, 3]],

       [[4, 6],
        [5, 7]]])
```

```
np.swapaxes(c,1,2)
```

```
array([[[0, 2],
        [1, 3]],

       [[4, 6],
        [5, 7]]])
```

NumPy Arithmetic Operation

```
a=np.arange(9).reshape(3,3)
b=np.array([10,100,1000])
print('add:\n',np.add(a,b))
print('substract:\n',np.subtract(a,b))
print('multiply:\n',np.multiply(a,b))
print('divide:\n',np.divide(a,b))
```

```
add:
 [[  10  101 1002]
 [  13  104 1005]
 [  16  107 1008]]
substract:
 [[ -10  -99 -998]
 [  -7  -96 -995]
 [  -4  -93 -992]]
multiply:
 [[   0  100 2000]
 [  30  400 5000]
 [  60  700 8000]]
divide:
 [[0.    0.01  0.002]
 [0.3   0.04  0.005]
 [0.6   0.07  0.008]]
```

Slicing

```
a=np.arange(20)
a

array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15,
16,
       17, 18, 19])

print(a[4:12:2])
s=slice(2,20,3)
print(a[s])

[ 4  6  8 10]
[ 2  5  8 11 14 17]
```

Iteration over Array

```
a=np.arange(0,48,4)
print(a)
a =a.reshape(3,4)
print(a)
for x in np.nditer(a):
    print(x)

[ 0  4  8 12 16 20 24 28 32 36 40 44]
[[ 0  4  8 12]
 [16 20 24 28]
 [32 36 40 44]]
0
4
8
12
```

```
16
20
24
28
32
36
40
44
```

Iteration order (c-style and f-style)

```
print("for c type")
for x in np.nditer(a, order="c"):
    print(x)
print("for f type")
for x in np.nditer(a,order="F"):
    print(x)
```

```
for c type
0
4
8
12
16
20
24
28
32
36
40
44
for f type
0
16
32
4
20
36
8
24
40
12
28
44
```

Joining Arrays

```
a=np.array([[1,2],[2,4]])
print('first array')
print(a)
```

```
b=np.array([[5,6],[10,14]])
print('second array')
print(b)
print('joining them along axis 0:')
print(np.concatenate((a,b)))
print('joining them along axis 1:')
print(np.concatenate((a,b),axis=1))
```

```
first array
[[1 2]
 [2 4]]
second array
[[ 5  6]
 [10 14]]
joining them along axis 0:
[[ 1  2]
 [ 2  4]
 [ 5  6]
 [10 14]]
joining them along axis 1:
[[ 1  2  5  6]
 [ 2  4 10 14]]
```

Splitting srray

```
a= np.arange(0,100,12)
print(a)
print(np.split(a,3))
print(np.split(a,[2,5])) #it will split on the postions given
```

```
[ 0 12 24 36 48 60 72 84 96]
[array([ 0, 12, 24]), array([36, 48, 60]), array([72, 84, 96])]
[array([ 0, 12]), array([24, 36, 48]), array([60, 72, 84, 96])]
```

resizing the array

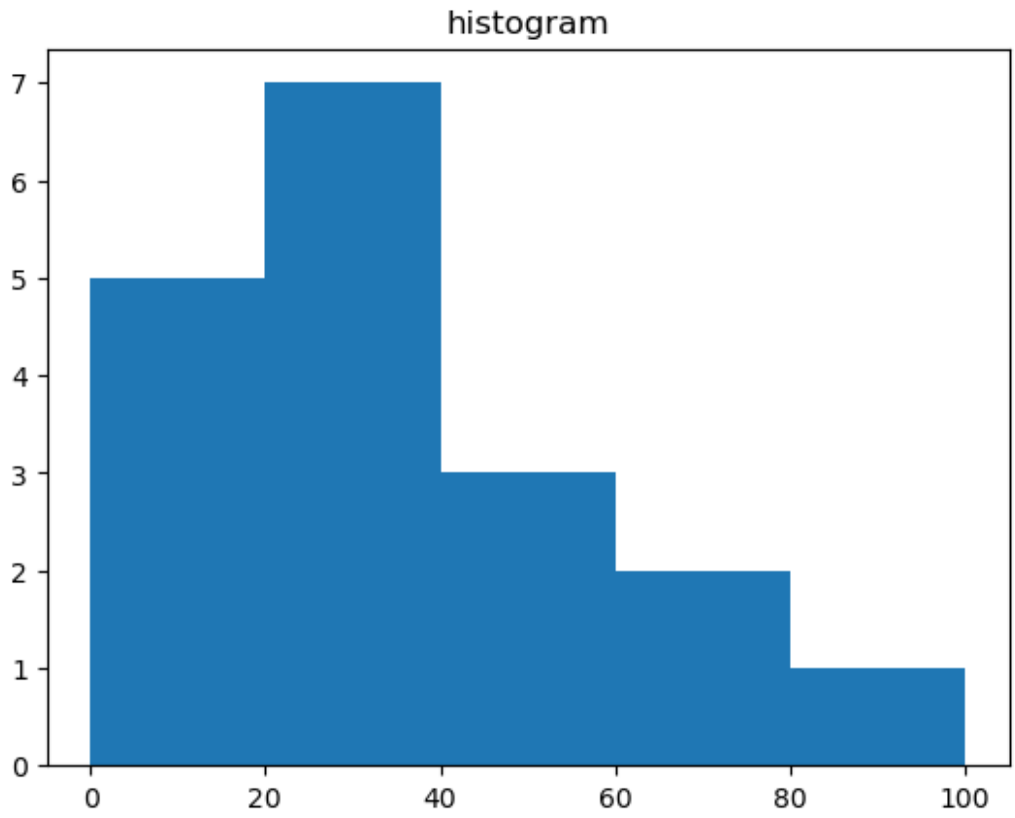
```
a= np.arange(4,52,3)
print(a)
print(a.shape)
a= np.resize(a,(4,5)) #if there is not enough value it will repeat
from the start
print(a)
print(a.shape)
```

```
[ 4  7 10 13 16 19 22 25 28 31 34 37 40 43 46 49]
(16,)
[[ 4  7 10 13 16]
 [19 22 25 28 31]
 [34 37 40 43 46]
```

```
 [49  4  7 10 13]]
(4, 5)
```

Histogram

```
from matplotlib import pyplot as plt
a=np.array([20,12,34,53,23,12,20,20,12,34,45,67,73,22,12,97,6,56])
plt.hist(a,bins=[0,20,40,60,80,100])
plt.title('histogram')
plt.show()
```



Other useful functions in numpy

```
#inspace fuction
a= np.linspace(1,3,12)
print(a)
```

```
[1.         1.18181818 1.36363636 1.54545455 1.72727273 1.90909091
 2.09090909 2.27272727 2.45454545 2.63636364 2.81818182 3.        ]
```

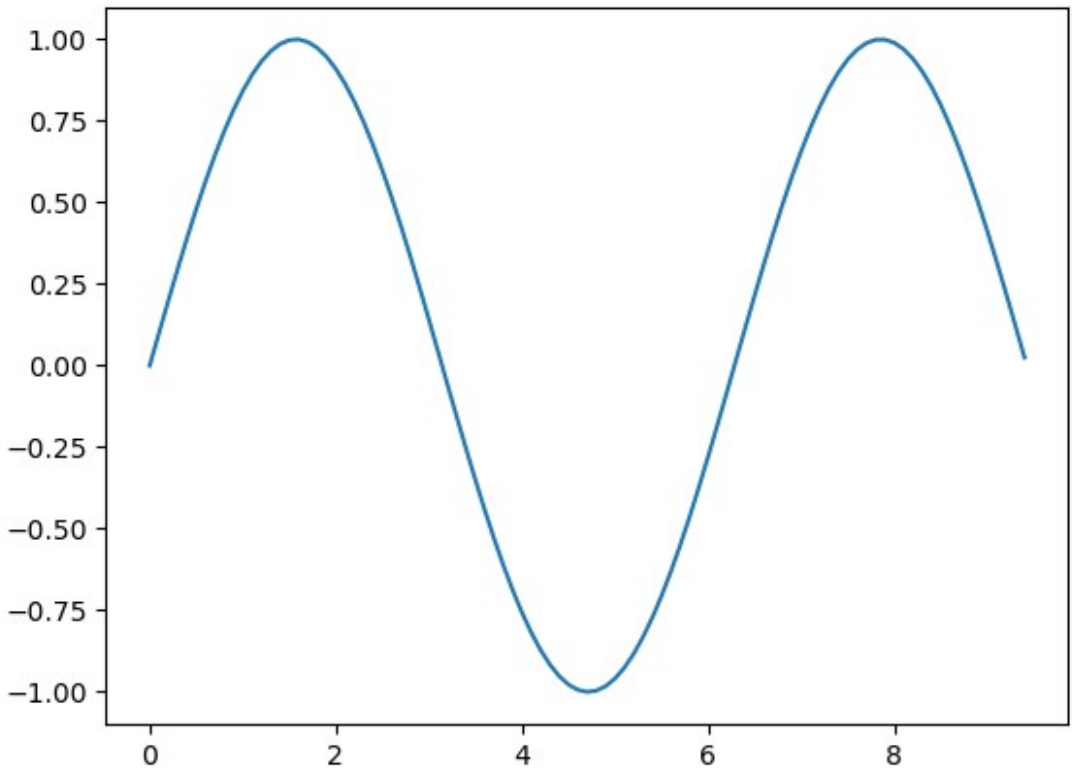
```
# sum and axis
a= np.array([(1,2,3,4),(5,6,7,8)])
print(a.sum(axis=0))
print(a.sum(axis=1))
```

```
[ 6  8 10 12]
[10 26]

#square root and standard deviation
print(np.sqrt(a))
print(np.std(a))

[[1.         1.41421356 1.73205081 2.        ]
 [2.23606798 2.44948974 2.64575131 2.82842712]]
2.29128784747792

#ravel function
a= np.array([(1,2,13,4),(5,62,17,8)])
print(a.ravel())

[ 1  2 13  4  5 62 17  8]
```

Fun Questions

```
#Now plotting sin graph
x=np.arange(0,3*np.pi,0.1)
y=np.sin(x)
#z=np.arange(0,3)
print(x)
print(y)
plt.plot(x,y) #plt.plot(x,y,z)
plt.show()

[0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.  1.1 1.2 1.3 1.4 1.5 1.6
1.7
 1.8 1.9 2.  2.1 2.2 2.3 2.4 2.5 2.6 2.7 2.8 2.9 3.  3.1 3.2 3.3 3.4
3.5
 3.6 3.7 3.8 3.9 4.  4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8 4.9 5.  5.1 5.2
5.3
 5.4 5.5 5.6 5.7 5.8 5.9 6.  6.1 6.2 6.3 6.4 6.5 6.6 6.7 6.8 6.9 7.
7.1
 7.2 7.3 7.4 7.5 7.6 7.7 7.8 7.9 8.  8.1 8.2 8.3 8.4 8.5 8.6 8.7 8.8
8.9
 9.  9.1 9.2 9.3 9.4]
[ 0.          0.09983342  0.19866933  0.29552021  0.38941834
0.47942554
  0.56464247  0.64421769  0.71735609  0.78332691  0.84147098
0.89120736
  0.93203909  0.96355819  0.98544973  0.99749499  0.9995736
0.99166481
  0.97384763  0.94630009  0.90929743  0.86320937  0.8084964
0.74570521
  0.67546318  0.59847214  0.51550137  0.42737988  0.33498815
0.23924933
  0.14112001  0.04158066 -0.05837414 -0.15774569 -0.2555411  -
```

```
0.35078323
 -0.44252044 -0.52983614 -0.61185789 -0.68776616 -0.7568025  -
0.81827711
 -0.87157577 -0.91616594 -0.95160207 -0.97753012 -0.993691   -
0.99992326
 -0.99616461 -0.98245261 -0.95892427 -0.92581468 -0.88345466 -
0.83226744
 -0.77276449 -0.70554033 -0.63126664 -0.55068554 -0.46460218 -
0.37387666
 -0.2794155  -0.1821625  -0.0830894   0.0168139   0.1165492
0.21511999
  0.31154136  0.40484992  0.49411335  0.57843976  0.6569866
0.72896904
  0.79366786  0.85043662  0.8987081   0.93799998  0.96791967
0.98816823
  0.99854335  0.99894134  0.98935825  0.96988981  0.94073056
0.90217183
  0.85459891  0.79848711  0.7343971   0.66296923  0.58491719
0.50102086
  0.41211849  0.31909836  0.22288991  0.12445442  0.02477543]
```




```
#create a 5*5 two-dimention array,and let 1 and 0 be placed
alternatively across the diagonals
a=np.zeros((5,5),dtype=int)
a[1::2,::2]=1
```

```
print(a)
print('\n')
a[::2,1::2]=1
print(a)
```

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


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

```
h = np.random.rand(5,5)
h[np.random.randint(5, size=3), np.random.randint(5, size=3)] = np.nan
h
```

```
array([[0.42955827, 0.53175444, 0.62714749, 0.11727943, 0.51253656],
       [0.31476353, 0.03763484, 0.03469238,        nan, 0.14410527],
       [       nan, 0.74063499, 0.4230139 , 0.00981315, 0.21978306],
       [0.27853654, 0.30760684, 0.37984629, 0.44983677,        nan],
       [0.11930888, 0.47496615, 0.02216922, 0.62975074, 0.95919234]])
```

```
print("total no. of missing value:\n",np.isnan(h).sum())
print("the idexes which are missing value: \
n",np.argwhere(np.isnan(h)))
inds=np.where(np.isnan(h))
print(inds)
h[inds]=0
h
```

```
total no. of missing value:
 0
the idexes which are missing value:
 []
(array([], dtype=int64), array([], dtype=int64))
```

```
array([[0.42955827, 0.53175444, 0.62714749, 0.11727943, 0.51253656],
       [0.31476353, 0.03763484, 0.03469238, 0.        , 0.14410527],
       [0.        , 0.74063499, 0.4230139 , 0.00981315, 0.21978306],
       [0.27853654, 0.30760684, 0.37984629, 0.44983677, 0.        ],
       [0.11930888, 0.47496615, 0.02216922, 0.62975074, 0.95919234]])
```