

DAY-1

NumPy, which stands for Numerical Python, is a fundamental package for numerical computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently.

#1.To create numpy array

```
l1 = [1,2,3,4,5] arr1 =  
np.array(l1) print(arr1.dtype)  
int32  
object
```

#to create a range of numbers
import numpy as np
ar1 = np.arange(13)
print(ar1)
print(len(ar1))

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

#identity matrix can be created using 2 functions

#eye() --> used to define both userdefined rows and columns

#identity() --> used to define same no. of rows and columns

```
print(np.eye(5)) print()  
print(np.eye(3,4))
```

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

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

```
np.identity(5 ,dtype=complex)
```

```
array([[1.+0.j,          0.+0.j,          0.+0.j,          0.+0.j,          0
.+0.j],
       [0.+0.j,          1.+0.j,          0.+0.j,          0.+0.j,          0
.+0.j],
       [0.+0.j,          0.+0.j,          1.+0.j,          0.+0.j,          0
.+0.j],
       [0.+0.j,          0.+0.j,          0.+0.j,          1.+0.j,          0
.+0.j],
       [0.+0.j,          0.+0.j,          0.+0.j,          0.+0.j,          1
.+0.j]])
```

#to convert a 1d array to multidimentional array

```
new_ar =np.array([1,2,3,4,5,6]) new_ar.reshape(3,2)
```

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

#slicing in multidimentional array new_ar

=

```
np.array([[1,2,3,4,5],[11,22,33,44,55]])
new_ar[1, 1:4]
```

```
array([22, 33, 44])
```

#to perform multidimentional slicing#ar2[rows,colums,step]

#ar2[row_strat:row_end, col_start, col_end, step]

```
ar2 = np.arange(25).reshape(5,5)print(ar2) print()
```

```
print(ar2[1:3, 1:3]) print()
```

```
print(ar2[-4:-2, -4:-2])
```

```
[[ 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]]
```

[[6 7]
[11 12]]

#to perform mean, median, sum, variance =median/tot no.of elements and standard deviation = sqrt of variance new_ar = np.arange(9) new_ar.mean()
np.median(new_ar) sum(new_ar) np.sum(new_ar, axis=0) #to add based on
column wise --> axis=0 np.var(new_ar,axis=0) np.std(new_ar)

2.581988897471611

```
print(arr1) li  
= list(arr1) print(li) print(arr1.tolist())  
[1 2 3 4 5]  
[10 20 30 40 50]
```

```
print(type(arr1)) print(type(li))
```

```
<class 'numpy.ndarray'>  
<class 'list'>
```

#operations on the elements in array arr1*2 arr1+4 arr1/3 array([0.33333333,
0.66666667, 1. , 1.33333333, 1.66666667])

#converts all the elements to same datatype

```
arr = [1,2,3.45,9,89,2] np_ar =  
np.array(arr) np_ar array([ 1. , 2. ,3.45, 9. , 89. ,  
2. ])
```

#to add element to array new_ar = np.append(np_ar,78) new_ar array([1. , 2. , 3.45, 9. , 89. , 2. , 78.])

#to add multiple values to array new_ar =
np.append(np_ar, [56,90,89]) new_ar
array([1. , 2. , 3.45, 9. , 89. ,
2. , 56. , 90. , 89.])

#inserting element based on index value

```
new_ar = np.insert(np_ar, 3, 123)  
new_ar array([ 1. , 2. ,3.45,123. ,9. , 89. , 2. ])
```

#inserting multiple elements based on index value new_ar =
np.insert(np_ar,3, [456, 932,
189]) new_ar array([1. , 2. ,
3.45, 456. , 93
2. , 189. , 9. , 89. ,])

#delete element from array nr = np.delete(np_ar,
np.where(np_ar==2.0)) nr array([1. , 3.45, 9. , 89.])

#to delete no. of values in array nr =
np.setdiff1d(np_ar, [1.0,2.0,9.0]) nr
array([3.45, 89.])

#to delete based on index position

```
nr = np.delete(np_ar, 3) print(nr)
```

#to delete multiple values based on index position

```
nr = np.delete(np_ar, [1,2]) print(nr)
```

```
[ 1. 2. 3.45 89. 2. ]  
[ 1. 9. 89. 2.]
```

```
l2 = [1,45,23,90,78,12,94,26,15,8,7]
new_ar = np.array(l2) #Filter in numpy array
print(new_ar<30) print(new_ar[new_ar<30])
```

```
[  True False  True False False          True False
  True  True  True  True  True]
[  1 23 12 26 15          8    7]
```

```
#values < 20 and values > 50 print((new_ar>20) & (new_ar<50))
print(new_ar[(new_ar>20) & (new_ar<50)])
[False  True  True False False False False  True False False False]
[45 23 26]
```

```
#to replace value in array
new_ar[new_ar==23]=156 new_ar array([
1,    45, 156,    90,    78,    12,    94,
26,    15,    8,    7])
```

```
#to replace value in array
new_ar[new_ar==23]=156 new_ar array([
1,    45, 156,    90,    78,    12,    94,
26,    15,    8,    7])
```

```
#to create 1d array full of zeros
print(np.zeros(5))
#to create 2d array full of zerosprint("\n",np.zeros([3,5])) #to
create 1d array full of ones print("\n",np.ones(4)+1) #to
create 2d array full of ones print("\n",np.ones([3,5]))
```

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

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

```
[2. 2. 2. 2.]
```

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

#type convert the element in array #to find numpy datatype of element --> dtype

```
ar = np.ones([3,4]) print(ar.dtype) ar = np.ones([3,4], dtype=int)
print(ar) print(ar.dtype)
```

```
float64 [[1 1 1 1]
         [1 1 1 1]
         [1 1 1 1]]
int32
```

#type convert the elements into various bits --> int16, int32, complex64... #changing to float

```
ar = np.zeros([2,4], dtype="float32") print(ar)
```

#changing again into int

```
print("\n", np.int16(ar))
```

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

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

#to print evenly separated points b/w 2 range of values --> linspace(start, end, no. of values) np.linspace(1,10,5) np.linspace(1,10,5, retstep=True)
#show the difference --> retstep()

```
(array([ 1.    ,  3.25,  5.5 ,  7.75, 10.
        ]), 2.25)
```

#random module #np.random.randint(start, end)

```
np.random.randint(20, 30)
```

```
23
```

#rand() --> to get randomly values from 0 to 1, based on uniform distribution

```
np.random.rand(4) #normalized distribution np.random.randn(5,4)
```

```
array([[ 1.04561362,  0.16759046, -
 0.4104      8036,      1.39404418],
       [ 0.38661777,          -0.10754625,      0.6068
```

```

4145,      1.15756147],
          [-0.50257716,      -1.85257614,      -0.2642
2393,      0.2872465 ],
          [-1.21715628,      2.65305324,      1.4108
8019,      0.03296527],
          [ 1.0091118 ,      -0.63340935,      -0.7245
0516,      -3.05591276]])

```

#min and max values

```
ar2.min() ar2.max()
```

#to find the index of min,max values ar2.argmin() 0 ar2.argmax()

#sin values ar3 =

```

np.arange(1,9)
np.sin(ar3) np.cos(ar3)/np.sin(ar3)
array([ 0.64209262, -0.45765755, -
7.01525 255,  0.86369115, -0.29581292,
-3.436353 ,  1.14751542, -0.14706
506])

```

#to perform (1x4)+(2x5)+(3x6)-->32

```

np2 = np.array([1,2,3]) np3 = np.array([4,5,6]) pro = np2*np3
sum(pro) np.dot(np2,np3)
32

```

#replacing empty element values with 1

```

np2 = np.array([1,2,3,9,10]) np3 = np.array([4,5,6]) length = len(np2)-
len(np3) for i in range(length):
np3 = np.append(np3,1) pro = np2*np3sum(pro)
51

```

#to replace value in array

```

new_ar[new_ar==23]=156 new_ar array([
1,      45, 156,      90,      78,      12,      94,
26,      15,      8,      7])

```

#to create 1d array full of zeros

```
print(np.zeros(5))
```

#to create 2d array full of zeros print("\n",np.zeros([3,5])) **#to**

create 1d array full of ones print("\n",np.ones(4)+1) **#to**

create 2d array full of ones `print("\n",np.ones([3,5]))`

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

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

```
[2. 2. 2. 2.]
```

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

#type convert the element in array #to find numpy datatype of element --> dtype

```
ar = np.ones([3,4]) print(ar.dtype) ar = np.ones([3,4], dtype=int)  
print(ar) print(ar.dtype)
```

```
float64 [[1 1 1 1]  
         [1 1 1 1]  
         [1 1 1 1]]
```

```
int32
```

#type convert the elements into various bits --> int16, int32, complex64... #changing to float

```
ar = np.zeros([2,4], dtype="float32") print(ar)
```

#changing again into int

```
print("\n", np.int16(ar))
```

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

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

#to print evenly seperated points b/w 2 range of values --> linspace(start, end,no.of values) np.linspace(1,10,5) np.linspace(1,10,5, retstep=True)
#showsthe difference --> retstep()

```
(array([ 1.   ,  3.25,  5.5 ,  7.75, 10.
]), 2.25)
```

#random module #np.random.randint(start, end)

```
np.random.randint(20, 30)
```

```
23
```

#rand() --> to get randomly values from 0to 1, based on uniform distribution

```
np.random.rand(4) #normalized distribution np.random.randn(5,4)
```

```
array([[ 1.04561362,  0.16759046, -
0.4104      8036,      1.39404418],
       [ 0.38661777,                    -0.10754625,      0.6068
4145,      1.15756147],
       [-0.50257716,                    -1.85257614,      -0.2642
2393,      0.2872465 ],
       [-1.21715628,                    2.65305324,      1.4108
8019,      0.03296527],
       [ 1.0091118 ,                    -0.63340935,      -0.7245
0516,     -3.05591276]])
```

#min and max values

```
ar2.min() ar2.max()
```

#to find the index of min,max values ar2.argmin() ar2.argmax()

#sin values ar3 =

```
np.arange(1,9)
```

```
np.sin(ar3) np.cos(ar3)/np.sin(ar3)
```

```
array([ 0.64209262, -0.45765755, -
7.01525 255,  0.86369115, -0.29581292,
       -3.436353   ,  1.14751542, -0.14706
506])
```

#to perform (1x4)+(2x5)+(3x6)-->32

```
np2 = np.array([1,2,3]) np3 = np.array([4,5,6]) pro = np2*np3
sum(pro) np.dot(np2,np3)
```

32

#replacing empty element values with 1

```
np2 = np.array([1,2,3,9,10]) np3 = np.array([4,5,6]) length = len(np2)-  
len(np3) for i in range(length):  
np3 = np.append(np3,1) pro = np2*np3sum(pro)  
51
```

#inserting not a number value np1

=

```
np.array([1,34,56,78,np.nan,np.nan])  
print(np1[~np.isnan(np1)]) #removing nan  
[ 1. 34. 56. 78. nan nan]  
[ 1. 34. 56. 78.]
```

E.LIKHITHA SAI

22A81A0614

