# Class-02

## Numpy

* Import numpy as np
  from numpy import *
  np.add( )
  add( )

* from numpy import *
  v = array([1,2,3,4], dtype = float)  # 1 D Array
  Print (v)
  Print (type(v))

  out :- [1.2. 3. 4.]
  <class 'numpy.ndarray'>
  <class 'list'>

  x = [1,2,3,4]
  Print (type(x))

* from numpy import *
  M = array([1,2,3],[3,4,1],[5,6,2],[8,9,10])  # 2D array
  Print (M)

  Out :- [[1,2,3]
  [3.4,1]
  [5.6,2]
  [8.9,10]]

* z = array([1,2],[3,4])

  out : (2,2)
  [[1,2]
  [3 4]]

  Print (z.shape)
  Print (z)

* z = array([1,2,3])
  z.shape
  out (3,0)

* from numpy import *
  z = array([[[1,2,3,7],[3,8,4,5],[5,9,9,0],[9,9,9,9]],
  [[1,2,70],[0,3,4,8],[8,7,9,2],[1,2,9,8]]]

Print(z.shape)
Print(z.size)

out : (2, 4, 4)

Class-04 Dt- 03/04

## Using array-generating function

arange

```
x = arange (5,14,1)      # Out - ([5,6,7,8,9,10,11,12,13])
x
```

linspace & logspace

linear space — logarithmic space

```
from numpy import *
linspace (0,10,10)
```

Out:- array ([0. , 1.11111111 , 2.22222222, 3.33333333, 4.44444444,
5.55555556, 6.66666667, 7.77777778, 8.88888889, 10. ])

```
linspace (0, 100, 20)
```

out:- ....

```
logspace (0, 10, endpoint = True, base = 8)
```

out- array ([1.00000000e+00, 1. ---------- ]).

```
* x = (1,2,3)
  y = asarray (x, dtype = float)
  y
out - array ([1.,2., 3.])
```

```
* from numpy import *
  x, y = mgrid [2:4, 2:1]
  x
out:- array ([[2,2,2],
              [3,3,3]])
  y
out- array ([[2,3,4],
             [2,3,4]])
```

```
* z, m = mgrid [2:6, 1:3]
  z
out- array ([[2,2],
             [3,3],
             [4,4],
             [5,5]])
  m
out - array ([[1,2],
              [1,2],
              [1,2],
              [1,2]])
```

random data

* from numpy import *
random.randn(5,5)

out :-

* diag (diagonal matrix)

diag([1,2,3,4,5,6])

out- array([[1, 0, 0, 0, 0, 0],
[0 2 0 0 0 0],
[0 0 3 0 0 0],
[0 0 0 4 0 0],
[0 0 0 0 5 0],
[0 0 0 0 0 6]])

diag([1,2,3])

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

* diag([1,2,3], K=1)

out : array([[0, 1, 0, 0],
[0, 0, 2, 0],
[0, 0, 0, 3],
[0, 0, 0, 0]])

Class-06 Dt- 05/05/2023

Zeros and ones

* zeros((3,3))

out: array([[0.,0.,0.],
[0.,0.,0.],
[0.,0.,0.]])

* Ones((4,3))

out: array([[1., 1., 1.],
[1., 1., 1.],
[1., 1., 1.],
[1., 1., 1.]])

More properties of the numpy arrays

M.itemsize # bytes per element

out: 8

M.nbytes #number of bytes

out: 72

M.ndim # number of dimensions

out : 2

# Manipulating arrays

## Indexing

```
* v[0]          # v is a vector, and thus only one dimension,
                 taking one index
out: 1

* m[1,1]        # m is a matrix, or a 2 dimensional array,
out: 0.47913739999   taking two indices

* m
out- array([[0.77872576, 0.40043577, 0.66254019],
            [0.60410063, 0.4791374,  0.8237106 ],
            [0.96856318, 0.15459644, 0.96082399]])

* m[1]          # row, column
out: array([0.60410063, 0.4791374, 0.8237106])

* m[1,:]        # row 1
out: array([0.60410063, 0.4791374, 0.8237106])

* m[:,1]        # column 1
out: array([0.40043577, 0.4791374, 0.15459644])

* m[0,0] = 1
  m
out: array([[1.        , 0.40043577, 0.66254019],
            [0.60410063, 0.4791374, 0.8237106 ],
            [0.96856318, 0.15459644, 0.96082399]])

* m[1,:] = 0
  m[:,2] = -1
  m
out: array([[1.        , 0.40043577, -1.        ],
            [0.        , 0.        , -1.        ],
            [0.96856318, 0.15459644, -1.        ]])
```

## Index Slicing

```
from numpy import *
A = array([1,2,3,4,5])
A
out- array([1,2,3,4,5])
```

- `A[2:6:4]`
  `out - array([3])`
- `A[1:3] = [-2,-3]`

change

```
A
out - array([1,-2,-3,4,5])
```

- `A[::1]`
  `out- array([1,-2,-3,4,5])`
- `A[::2]`
  `out- array([1,-3,5])`
- `A[:3]`
  `out- array([1,-2,-3])`
- `A[3:]`
  `out- array([4,5])`

- from numpy import *

```
A = array([n+m*10 for n in range(5)
                  for m in range(5)])
A
out: array([[ 0,  1,  2,  3,  4],
            [10, 11, 12, 13, 14],
            [20, 21, 22, 23, 24],
            [30, 31, 32, 33, 34],
            [40, 41, 42, 43, 44]])
```

* A[1:4, 1:4] # a block from the original array

out: array([[11, 12, 13],
[21, 22, 23],
[31, 32, 33]])

* A[::2, ::2]

out: array([[ 0, 2, 4],
[20, 22, 24],
[40, 42, 44]])

Fancy indexing

from numpy import *

B = array([n for n in range(5)])

B

out: array([0, 1, 2, 3, 4])

* row_mask = array([True, True, True, False, False])

B[row_mask]

out- array([0, 1, 2])

* row_mask = array([1, 1, 1, 0, 0])

B[row_mask]

out:- array([1. 1. 1. 0. 0.])

B

0, 1, 2, 3, 4

| | | | | |
|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 |
| 10 | 11 | 12 | 13 | 14 |
| 20 | 21 | 22 | 23 | 24 |
| 30 | 31 | 32 | 33 | 34 |

| | | | | |
|---|---|---|---|---|
| 0 | 1 | 4 | 9 | 16 |
| 100 | 121 | 144 | 169 | 19 |
| 400 | 441 | 484 | 529 | 5 |
| 900 | 961 | 1024 | 1089 | 11 |

Class-07 Dt-06/05/2023

```
from numpy import *
x = arrange(0, 10, 0.5)        out: ([0. , 0.5, 1. , 1.5, 2. ... 9.5])
x
* mask = (5<x) * (x<7.5)
mask
out- array([False, False, False, False, False, False, False, False, False,
            False, False, True, True, True, True, False, False, False,
            False, False], dtype=bool)
* x[mask]
out- array([5.5, 6. , 6.5, 7. ])
```

Where

```
indices = where(mask)
indices
out- (array([11, 12, 13, 14]),)
x[indices]
out- array([5.5, 6. , 6.5, 7. ])
```

diag

```
diag(A)
out: array([0, 11, 22, 33, 44])
diag(A, -1)
out: array([10, 21, 32, 43])
```

A-
```
[[ 0  1  2  3  4]
 [10 11 12 13 14]
 [20 21 22 23 24]
 [30 31 32 33 34]
 [40 41 42 43 44]]
```

take

```
* v2 = arrange(-3, 3)
v2
out- array([-3, -2, -1, 0, 1, 2])
* row_indices = [1, 3, 5]
v2[row_indices]
out: array([-2, 0, 2])
* v2.take(row_indices)
out- array([-2, 0, 2])
```

* take([-3,-2,-1,0,1,2], row_indices)
out- array([-2,0,2])

Choose

```
from numpy import *
which = [1,0,0,1]
Choices = [[-3,-2,-3,-2], [7,4,7,4]]
Choose(which, choices)
out:- array([7,-2,-3,4])
```

Linear algebra -

Scalar - array Operation :-

```
from numpy import *
v1 = arrange(0,5)
v
out - array([0,1,2,3,4])
```

* print(A)
A*2, A-2

```
[[0  1  2  3  4]
 [10 11 12 13 14]
 [20 21 22 23 24]
 [30 31 32 33 34]
 [40 41 42 43 44]]
```

out- (array([[[0, 2, 4, 6, 8],
[20, 22, 24, 26, 28],
[40, 42, 44, 46, 48],
[60, 62, 64, 66, 68],
[80, 82, 84, 86, 88]]),

array([[[-2, -1, 0, 1, 2],
[8, 9, 10, 11, 12],
[18, 19, 20, 21, 22],
[28, 29, 30, 31, 32],
[38, 39, 40, 41, 42]]))

* v1*2
out- array([0, 2, 4, 6, 8])

* v1+2
out- array([2, 3, 4, 5, 6])

* A*A

```
array([[  0,   1,   4,   9,  16],
       [100, 121, 144, 169, 196],
       [400, 441, 484, 529, 576],
       [900, 961, 1024, 1089, 1156],
       [1600, 1681, 1764, 1849, 1936]])
```

* Print (v1)
  v1 * v1
out- [0, 1, 2, 3, 4]
array([0, 1, 4, 9, 16])

* A. shape, v1. shape
((5,5), (5, ))

* A * v1

[[0 1 2 3 4]
[10 11 12 13 14]
[20 21 22 23 24]
[30 31 32 33 34]
[40 41 42 43 44]]

[0, 1, 2, 3, 4]

out- array([[[ 0, 1, 4, 9, 16],
[ 0, 11, 24, 39, 56],
[ 0, 21, 44, 69, 96],
[ 0, 31, 64, 99, 136],
[ 0, 41, 84, 129, 176]]])

## matrix algebra

* dot (A, A)

out: array([[ 300, 310, 320, 330, 340],
[1300, 1360, 1420, 1420, 1540],
[2300, 2410, 2520, 2630, 2740],
[3300, 3460, 3620, 3780, 3940],
[4300, 4510, 4720, 4930, 5140]])

* dot (A, v1)

out - array([30, 130, 230, 330, 430])

* m = matrix (A)
  Print (m)
  v = matrix (v1). T
  v

out- [[1 -2 -3 4 5]]

matrix([[0],
[1],
[2],
[3],
[4]])

* c = matrix ([1, 2, 3, 4, 5])
  c. T

out - matrix([[1],
[2],
[3],
[4],
[5]])

Class - 08 Dt - 07/05

## Array/Matrix transformation

```
c = matrix([[1-1.3j, 5+2j], [3j, 7+4j]])
```
real: 1. 3j imag

```
c
out - matrix([[1.-1.3j, 5.+2.j],
              [0.+3.j, 7.+4.j]])
```

# a+jb / ib -> imaginary part
a -> real part a
a, b -> real number

```
* Conjugate(c)
out - matrix([[1.+1.3j, 5.-2.j],
              [0.-3.j, 7.-4.j]])
```

# 1+1j -> 1-1j
3+3j -> 3-3j
j -> -j

```
* C.H
out - matrix([[0.-1.j, 0.-3.j],
              [0.-2.j, 0.-4.j]])
```

```
* real(c)
out - matrix([[1., 5.],
              [0., 7.]])
```

```
* imag(c)
matrix([[-1.3, 2.],
        [3., 4.]])
```

## Matrix computations

Inverse

```
*Linalg. inv(c)
out - matrix([[0.2+j, 0.-1.j],
              [0.-1.5j, 0.+0.5j]])
```

```
* C.I * C
out - matrix([[1.00000000e+00+0.j, 4.44089910e-16+0.j],
              [0.00000000e+00+0.j, 1.00000000e+00+0.j]])
```

Determinant

linalg.det(C)

out- (2.0000000000000004+0j)

linalg.det(C-I)

out-(0.5000000000000011+0j)

mode → 3

mean → 19/6 → 3.33

Median → 3+3/2 → 3

4 1 3 6 2 3

* mean(data[:3])

out- 6.197096947515854

Standard deviations and variance

std(data[:,3]), var(data[:,3])

out-(8.2822---, 68.596023---)

mix and max

data[:,3].min()

out- -25.800000000001

data[:,3].max()

out- 29.300000000000004

Dt - 10/05

# Class-09

## Reshaping, resizing & stacking

```
from numpy import *
A
```

out array([[0, 1, 2, 3, 4],
[10, 11, 12, 13, 14],
[20, 21, 22, 23, 24],
[30, 31, 32, 33, 34],
[40, 41, 42, 43, 44]])

```
* n, m = A.shape
* B = A.reshape((1, n*3))
B
```

out array([[0, 1, 2, 3, 4, 10, 11, 12, 13, 14, 20, 21, 22, 23, 24, 30,
31, 32, 33, 34, 40, 41, 42, 43, 44]])

```
* C = arange(0, 12)
C
d = C.reshape(4, 3)
d
```

out array([[0, 1, 2],
[3, 4, 5],
[6, 7, 8],
[9, 10, 11]])

```
* from numpy import *
b = random.rand(3, 3)
b[0, 0] = 12
Print(b)
```

out [[12. 0.804c7401 0.86135114]
[0.26140153 0.61689c14 0.01410927]
[0.64314837 0.85647648 0.35020965]]

```
* B[0, 0:5] = 5
B
```

out-array([[5, 5, 5, 5, 5, 10, 11, 12, 13, 14, 20, 21, 22, 23, 24, 30,
32, 33, 34, 40 41, 42, 43, 44]])

* A  # and the original variable is also changed.

Out: array([[5, 5, 5, 5, 5],
[10, 11, 12, 13, 14],
[20, 21, 22, 23, 24],
[30, 31, 32, 33, 34],
[40, 41, 42, 43, 44]])

* B = A.flatten()

B

Out: array([5, 5, 5, 5, 5, 10, 11, 12, 13, 14, 20, 21, 23, 24, 30,
31, 32, 33, 34, 40, 41, 42, 43, 44])

* B[0:5] = 10

B

Out: array([10, 10, 10, 10, 10, 10, 11, 12, 13, 14, 20, 21, 22, 23, 24,
30, 31, 32, 33, 34, 40, 41, 42, 43, 44])

* A  # now A has not changed, because B's data is a copy of A's.

Out: array([[5, 5, 5, 5, 5],
[10, 11, 12, 13, 14],
[20, 21, 22, 23, 24],
[30, 31, 32, 33, 34],
[40, 41, 42, 43, 44]])

Adding a new dimension : newaxis

* v = array([1, 2, 3])

v

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

* shape(v)

Out: (3,)

* v[:, newaxis]

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

* v[:, newaxis].shape
out- (3,1)

* v[newaxis,:].shap
out: (1,3)

## tile and repeat

* a = array([[1,2],[3,4]])

a
out- array([[1, 2],
[3, 4]])

* repeat(a,3)
out- array([1,1,1,2,2,2,3,3,3,4,4,4])

* ~~tile~~ tile(a,3)
out- array([[1,2,1,2,1,2],
[3,4,3,4,3,4]])

## Concatenate

* b= array([[5,6]])
b.T
out- array([[5],
[6]])

* Concatenate((a,b), axis=0)
out- array([[1,2],
[3,4],
[5,6]])

* Concatenate((a,b.T), axis=1)
Out- array([1,2,5],
[3,4,6]])

* b
out- array([[5,6]])

## hstack & vstack

* vstack((a, b))

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

* hstack((a, b.T))

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

## Copy and "deep copy"

* A = array([[1, 2], [3, 4]])

A

Out array([[1, 2],
[3, 4]])

* B = A

* B[0, 0] = 10

B

Out: array([[10, 2],
[3, 4]])

* A

Out array([[10, 2],
[2, 4]])

* B = copy(A)

* B[0, 0] = -5

B

Out array([[-5, 2],
[3, 4]])

* A

array([[10, 2],
[3, 4]])

Class - 10

Dt - 12/01

* v = array([1, 2, 3, 4])
  for i in v:
      Print(i)
  out- 1 2 3 4

* M = array([[1, 2, 2], [3, 2, 4]])
  for row in M:
      Print("row", row)
      for element in row:
          Print(element)
  out: row [1 2 2]
  1
  2
  2
  row [3 2 4]
  3
  2
  4

* for row_idx, row in enumerate(M):
      Print("row_idx", row_idx, "row", row) ~~[3,4]~~
      for col_idx, element in enumerate(row):
          Print("col_idx", col_idx, "element", element)
          # Update the matrix M: square each element
          M[row_idx, col_idx] = element **2

out: ('row_idx', 0, 'row', array([1, 2]))
('col_idx', 0, 'element', 1)
('col_idx', 0, 'element', 2)
('row_idx', 0, 'row', array([3, 4]))
('col_idx', 0, 'element', 3)
('col_idx', 1, 'element', 4)

* M

out- array([[1, 4],
[9, 16]])

Class - 11 Dt 13/01

# Broadcasting operation in case of numpy arrays

```
* from numpy import *
x = array([5, 8, 9, 10, 0])
y = array([5, 3, 2, 1, 9])
x+y
```

Out - array([10, 11, 11, 10, 9])

```
* from numpy import *
x = array([[0,0,0],[10,10,10],[50,50,50],[70,70,70],dtype=int)
y = array([[3.0, 4.0, 5.0]])
x+y
```

Out array([[ 3., 4., 5.],
[13., 14., 15.],
[53., 54., 55.],
[73., 74., 75.]])

```
* x = arange(0, 100, 5)
x = x.reshape(4,5)
Print(x)
z = x.T
print(z)
for i in z:
    for j in i:
        print(j)
```

Out:
```
[[ 0  5 10 15 20]
 [25 30 35 40 45]
 [50 55 60 65 70]
 [75 80 85 90 95]]

[[ 0 25 50 75]
 [ 5 30 55 80]
 [10 35 60 85]
 [15 40 65 90]
 [20 45 70 95]]
```

0 60
25 85
50 15
75 40
5 65
30 90
55 20
80 45
10 70
35 95

* for i in range (0,4):
  Print (x[i, 0:4])

out [0 5 10 15]
[25 30 35 40]
[50 55 60 65]
[75 80 85 90]

* x = arange (8). reshape (2, 4)
  Print (x)
  x. flatten (order = 'F')

out - [[0 1 2 3]
[4 5 6 7]]
array ([0, 4, 1, 5, 2, 6, 3, 7])

* x. flat [5]

out - 5

* y = arange (4). reshape (1, 4, 1)
  Print (y)
  rollaxis (y, 0, 2)

out - [[[0]
[1]
[2]
[3]]]
array ([[[0]],
[[1]],
[[2]],
[[3]]])

* from numpy import *
  y = arange (8). reshape (2, 2, 2)
  Print (y)
  rollaxis (y, 2, 2)

out - [[[0 1]
[2 3]]
[[4 5]
[6 7]]]
array([[[0, 1]
[2, 3]]
[[4, 5]
[6, 7]]])

```
* from numpy import *
z = arange(12)
Print(z)
y = splet(z,4)
x = splet(z, [0,6])
x
```

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

~~`[array([], d type =`~~

`[array([0, 1, 2]), array([3,4,5]), array([6, 7,8,9,10,11])]`

```
* z = arange(32). reshape(8,4)
Print(z)
y = splet(z,8)
y
```

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]
 [24 25 26 27]
 [28 29 30 31]]
```

```
[array([[0, 1, 2, 3]]),
 array([[4, 5, 6, 7]]),
 array([[8, 9, 10, 11]]),
 array([[12, 13, 14, 15]]),
 array([[16, 17, 18, 19]]),
 array([[20, 21, 22, 23]]),
 array([[24, 25, 26, 27]]),
 array([[28, 29, 30, 31]])]
```

```
* right_shift(8,2)
```

Out- 2

```
* z = array([0, 30, 8, 45, 60.2, 90])
cos(z* pi /180)
tan(z* pi / 180)
```

```
array([0.00000000e+00, 5.96119460e-01, 1.00000000e+00,
       1.74609743e+00, 1.63312394e+16])
```

```
* z = 4.2
  print (ceil(z))
  floor(z)
out 5.0
    4.0

* reciprocal(z)
out- 0.23809523809523808

* z = array ([0, 30.8, 45, 60.2, 90])
  y = array ([0, -30.8, 45, 60.8, 90])
  mod(z, y)
out - array([nan, -0., 0., 60.2, 0.])

* power (z, 2)
out - array([ 0. , 948.64, 2025. , 3624.04, 8100. ])

* z = array([[[6,8,2],[7,3,9],[2,0,7]])
  z
out - array([[[6, 8, 2],
              [7, 3, 9]
              [2, 0, 7]]])

* amin (z, axis = 1)
out array([2, 3, 0])
* amin (z, axis = 0)
out array([2,0,2])
* amax (z)
out - 9
```

```
* median (z)
* median (z,0)
* media (z,1)
* mean(z)
* mean (z,1)
* mean (z,0)
* std (z)
```