In: input code
Out: Output

Numpy

```
In : import mumpy as np
In : myarr = np.array([3,6,32,7]),np.int8)
In : myarr
Out : array([3,6,32,7]), dtype=int8)
In : myarr[0]
Out : 3
In myarr.shape
Out : (1,4)
In : myarr.dtype
Out : dtype('int8)
In : myarr[0] = 45
In : myarr
Out : array([45, 6, 32, 7])
```

ex of array being formed from a list

Array creation methods

① Conversion from other Py structures

```
In : List array = np.array([[1,2,3],[5,8,5],
                            [0,3,1]])
In : List array
Out : array([[1,2,3],
             [5,8,5],
             [0,3,1]])
In : List array.shape      Out : (2,3)
In : List array.size
Out : 9
```

In : np.array({34,23,23}) ⟷ array from set

Out: array({34, 23})

In : ~~np.~~ zeroes = np.zeros((2,5))

In : zeroes

↳ gives an array of zeros (2,5)

In : zeroes.shape

Out : (2,5)

In : zeroes.size   Out : 10

In : range = np.arange(15) → creats an array of (0 to n-1)

In : range

Out : array([0,1,2,3, -- 14])

In : Linspace = np.Linspace(1,5,12)

In : linspace

Out : gives an np array of 12 elements equally spaced b/w 1 & 5

In : emp = np.empty((4,6))

In : emp

Out : Creates a random array of (4,6)

In : emp_like = np.empty_like(linspace)

↳ creats an empty array same in size as the input array

In : ide = np.identity(10)

In : ide

Output : sq. matrix of I

In : arr = np.arange(99)

↳ 99×1 ≡ 3×33

In : arr

In : arr.reshape(3, 33)

In : arr ↳ forms a 3D array

In : arr = arr.ravel()

↳ back to normal

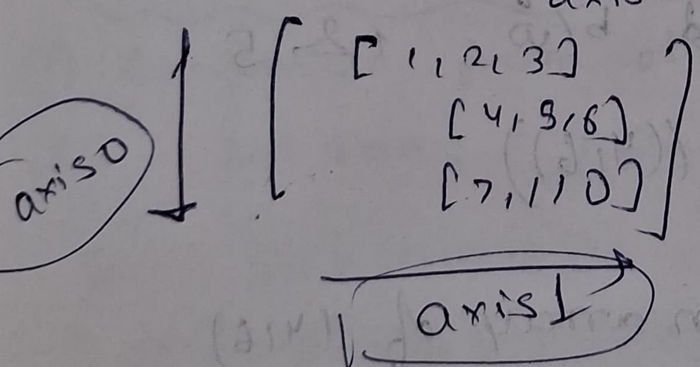
arr.reshape(3,33)
↳ forms a copy of arr

arrA = arr.reshape
↳ original (3,33) is updated

Numpy Axis

1D : [1, 2, 3, 4, 5] → 1 axis [Axis 0]

2D → ↳ 2 axis [Axis 0, Axis 1]



→ along row or cutting column : Axis 1

→ along column or cutting row : Axis 0

In : x = np.a

In : x = [[1, 2, 3], [4, 5, 6], [7, 1, 0]]

In : ar = np.array(x)

with ( ) : methods
without ( ) : attributes

```
In  : ar.sum (axis =0)
Out : array ([12, 8, 9])
In  : ar. sum (axis= 1)
Out : array ([6, 15, 9])
In  : ar. T   ↳ transpose of n-D array
In  : ar. flat  → gives an iterator
Out : <numpy.flatiter at Address>
In  : for item in ar.flat :
          print (item)
In  : ar. ndim
Out : no. of dimensions
In  : ar. size        Out : 9
In  : ar.nbytes       Out : 36 (9x4)
In  : one = np.array ([1, 3, 4, 634, 2])
In  : one. arg max ( )
Out : 3 ↳ gives index of max element
In  : one. arg min ( )
Out : 0
In  : one. arg sort ( ) → gives (index) of elements in sorted way
Out : [1, 2, 3, 4, 634]
```

```
ar = [ [1,2,3]
       [4,5,6]
       [7,1,0] ]
ar.argmin()          Out : 8
```

first conversts it into 1D

```
In : ar.argsort()
```

Out : a 2D array conteining indecis of elements in sorted manner

```
In : ar.argmax(axis=0)
Out : array([1, 2, 2])
In : ar.argmin(axis = 1)
Out : array([0, 0, 2])
In : ar.reshape(3x3 or (9,1), or (1,9))
```

Arethematic operations

```
ar = [[1, 2, 3]          ar2 = [[1,2,1]
      [4, 5, 6]                 [4,0,6]
      [7, 1, 0]                 [8,1,0]]
In : ar + ar2
Out = [[2, 4, 4],
       [8, 5, 12],
       [15, 2, 0]]
```

(this cannot happen without np

eg : [1,2] + [3,4] ≠ [1,2,3,4]

In : ar * ar 2   (ar - ar 2)

Out : array ([[1 , 4, 3],
              [16 , 0, 36],
              [5 6 , 1, 0]])

In: np. sqrt (ar)

Out - ↳ sq root of all elements

In : ar. sum()

Out : 29

In [: ar. max ()   Out : 7   } returns index

In : ar. min ()   Out : 0

In : np .where (ar>5) → write condition

↳ returns coor of elements which are > 5

In : np. count _ nonzero (ar)

Out : 8

In np.nonzero(ar) for each axis

↳ returns tuple of index with non zero elements are present

Searching : arr = np. array ([1, 2, 3, 4, 5, 6]))

x = np. where (arr == 4)

↓

returns the index where 4 occurred

Numpy consumes less space

```
In : import sys

In : py-ar = [ 0, 4, 55, 2]

In :  np-ar = np.array(py-ar)

In :  sys.getsizeof(1)* len(py_ar)

out :  56

In[ ] :  np-arr.itemsize * np_ar.size

Out :  16

In :  np-ar.tolist[ ]
Out :    [0, 4, 55, 2] → returns a python list of
                                 np
                              an array
```

(concatenation)

```
arr1 = np.array([1,2,3])
arr2 = np.array([4,5,6])
  arr = np.concatenate((arr1,arr2))
```

---

spliting

```
arr = np.array([1,2,3,4,5,6])
new arr = ~~arr~~ np.array-split (arr,3)
                                  ↓
                            array([1,2,3])
```