NUMPY BASICS

Importing numpy

In [20]:
```
import numpy as np
```

Creating ndarrays :

- array
- arange
- ones
- ones_like
- zeros
- zeros_like
- empty
- empty_like
- full
- full_like
- eye,identity

In [79]:

```
# np.array - converts the input collection to array

a = np.array([1,4,2,5,6,30])
print(type(a))

a = np.array((1,4,2,5,6,30))
print(type(a))


a = np.array({1,4,2,4,1,5,6,30})
print(type(a))


a = np.array("abcd")
print(type(a))


a = np.array({1:4,2:5,6:30})
print(type(a))

a = np.array(True)
print(type(a))

a = np.array(124)
print(type(a))

a = np.array([1,4,2,5,6,30], dtype='float64') # dtype=np.float64
print(a)
```

```
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
[ 1.  4.  2.  5.  6. 30.]
```

Lavanya | Value Laden

In [58]:

```python
# np.arange - returns range of elements in ndarray

a = np.arange(10)
print(a)

a = np.arange(2,5)
print(a)

a = np.arange(2,10,3)
print(a)

a = np.arange(5.,20.)
print(a)

a = np.arange(5.,20., dtype='int32')
print(a)
```

```
[0 1 2 3 4 5 6 7 8 9]
[2 3 4]
[2 5 8]
[ 5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17. 18. 19.]
[ 5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]
```

In [78]:

```python
# np.ones - returns 1D array with value, 1.

a = np.ones(5)
print(a)

a = np.ones(5,dtype='int32')
print(a)

a = np.arange(5)
print(a)

a = np.ones(a)
print(a)

b = np.ones(10)
print(b)
```

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

In [76]:

```
# np.ones_like - takes another array and produces ones array


a = np.arange(5)
print(a)

a = np.ones_like(a)
print(a)

b = np.ones_like(10)
print(b)
```

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

In [85]:

```
# np.zeros - produces an array with value 0.

a = np.zeros(10)
print(a)
print()

a = np.zeros((2,3))
print(a)
print()

a = np.zeros((2,3,2))
print(a)
print()

a = np.arange(5)
print(a)

a = np.zeros(a)
print(a)
```

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

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

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

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

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

In [87]:

```
# np.zeros_like - takes another array and produces ones array

a = np.arange(5)
print(a)

a = np.zeros_like(a)
print(a)
```

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

In [95]:

```
# np.empty - default array with memory is created

a = np.empty(10)
print(a)
```

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

In [100]:

```
# np.empty_like - new array is created with respect to array passed

a = np.arange(5)
print(a)

a = np.empty_like(a)
print(a)
```

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

In [103]:

```
# np.full - forms the array with fill_value

a = np.full(5, fill_value=10, dtype='float64')
print(a)
```

```
[10. 10. 10. 10. 10.]
```

In [105]:

```
# np.full_like - fills the array values with fill_value


a = np.arange(5)
print(a)

a = np.full_like(a, fill_value=5)
print(a)
```

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

Lavanya | Value Laden

In [107]:

```
# np.eye, np.identity - forms an n*n array with 1 on diagonal and 0 elsewhere

a = np.eye(4,dtype='int32')
print(a)

a = np.identity(4,dtype='int32')
print(a)
```

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

Basic array information

In [21]:

```
d = np.random.randn(2,3)
d
```

Out[21]:

```
array([[ 0.70470063, -0.03795867, -1.72136895],
       [ 0.09561663,  0.43604671, -0.4098891 ]])
```

In [41]:

```
type(d)
```

Out[41]:

```
numpy.ndarray
```

In [22]:

```
d.shape
```

Out[22]:

```
(2, 3)
```

In [165]:

```
a = np.arange(10).reshape(2,5)
a
```

Out[165]:

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

In [23]:

```
d.dtype
```

Out[23]:

```
dtype('float64')
```

In [24]:

```
d.ndim
```

Out[24]:

```
2
```

Array Data Types

- int8, uint8 : i1, u1
- int16, uint16 : i2, u2
- int32, uint32 : i4, u4
- int64, uint64 : i8, u8

- float16 : f2
- float32 : f4 or f
- float64 : f8 or d
- float128 : f16 or g
- complex64, complex128, complex256 : c8, c16, c32
- bool : ?
- object : 0
- string_ : S
- unicode_ : U

```
In [114]: a = np.arange(5, dtype='i4') # dtype='int32'
          print(a)

          [0 1 2 3 4]
```

```
In [113]: a = a.astype(np.float64)
          print(a.dtype)
          print(a)

          float64
          [0. 1. 2. 3. 4.]
```

Arithmetic Operations on arrays

```
In [115]: a = np.arange(5)
          print(a)

          [0 1 2 3 4]
```

```
In [116]: a*a

Out[116]: array([ 0,  1,  4,  9, 16])
```

```
In [117]: a*5

Out[117]: array([ 0,  5, 10, 15, 20])
```

```
In [118]: a-a

Out[118]: array([0, 0, 0, 0, 0])
```

```
In [119]: a-2

Out[119]: array([-2, -1,  0,  1,  2])
```

```
In [120]: a/2

Out[120]: array([0. , 0.5, 1. , 1.5, 2. ])
```

```
In [121]: a//2

Out[121]: array([0, 0, 1, 1, 2], dtype=int32)

In [122]: a**2

Out[122]: array([ 0,  1,  4,  9, 16], dtype=int32)

In [135]: a%2

Out[135]: array([0, 1, 0, 1, 0], dtype=int32)

In [123]: a>2

Out[123]: array([False, False, False,  True,  True])

In [124]: a==1

Out[124]: array([False,  True, False, False, False])

In [125]: a&1

Out[125]: array([0, 1, 0, 1, 0], dtype=int32)

In [126]: a^2

Out[126]: array([2, 3, 0, 1, 6], dtype=int32)
```

Basic Indexing and Slicing

```
In [136]: a = np.arange(10)
          print(a)

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

In [137]: a[2:]

Out[137]: array([2, 3, 4, 5, 6, 7, 8, 9])

In [138]: a[:5]

Out[138]: array([0, 1, 2, 3, 4])

In [139]: a[:]

Out[139]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Lavanya | Value Laden

```
In [140]: a[2:6:2]

Out[140]: array([2, 4])
```

```
In [141]: a[::-1]

Out[141]: array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
```

```
In [142]: a[-1]

Out[142]: 9
```

```
In [143]: a[3]

Out[143]: 3
```

```
In [144]: l = [[1,2,3],[4,5,6]]
          a = np.array(l)
          a[1][0]

Out[144]: 4
```

```
In [145]: a[1,0]

Out[145]: 4
```

Condition checks

```
In [176]: # values on conditions

          a = np.arange(10)
          print(a)

          print(a[a>4])

          print(a[~(a>4)])

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

```
In [177]: # get elements separate based on condition

          m = (a==1) | (a!=4)
          print(a[m])

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

Array functions/ working with arrays

- Unary functions :
    - abs, fabs
    - sqrt
    - square
    - exp
    - log, log10, log2, log1p
    - sign
    - ceil
    - floor
    - rint
    - modf
    - isnan
    - isfinite, isinf
    - cos, cosh, sin, sinh, tan, tanh
    - arccos, arccosh, arcsin, arcsinh, arctan, arctanh
    - logical_not
- Binary functions:
    - add, subtract, multiply, divide, floor_divide
    - power
    - maximum, fmax, minimum, fmin (fmax and fmin ignores NaN)
    - mod
    - copysign
    - greater, greater_equal, less, less_equal
    - logical_and, logical_or, logical_xor

In [181]:

```
# Transpose of array

a = np.random.randn(3,3)
print(a)
print()

print(a.T)
print()
```

```
[[-0.31368957  0.3635272   1.63620372]
 [ 1.66469763  1.77701997 -0.93083396]
 [-0.40271301 -0.31044093  1.29735744]]

[[-0.31368957  1.66469763 -0.40271301]
 [ 0.3635272   1.77701997 -0.31044093]
 [ 1.63620372 -0.93083396  1.29735744]]
```

Lavanya | Value Laden

In [184]:
```
# matrix multiplication

b = np.dot(a,a.T)
print(b)
```

```
[[ 2.90771579 -1.39923717  2.13621421]
 [-1.39923717  6.79547002 -2.42967948]
 [ 2.13621421 -2.42967948  1.94168766]]
```

In [185]:
```
a = np.arange(16).reshape(2,2,4)
a
```

Out[185]:
```
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7]],

       [[ 8,  9, 10, 11],
        [12, 13, 14, 15]]])
```

In [187]:
```
a.transpose((1,0,2))
```

Out[187]:
```
array([[[ 0,  1,  2,  3],
        [ 8,  9, 10, 11]],

       [[ 4,  5,  6,  7],
        [12, 13, 14, 15]]])
```

In [188]:
```
a.swapaxes(1,2)
```

Out[188]:
```
array([[[ 0,  4],
        [ 1,  5],
        [ 2,  6],
        [ 3,  7]],

       [[ 8, 12],
        [ 9, 13],
        [10, 14],
        [11, 15]]])
```

In [190]:
```
a = np.arange(5)
print(a)
```

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

In [191]:
```
np.sqrt(a)
```

Out[191]:
```
array([0.        , 1.        , 1.41421356, 1.73205081, 2.        ])
```

In [192]:
```
np.exp(a)
```

Out[192]:
```
array([ 1.        ,  2.71828183,  7.3890561 , 20.08553692, 54.59815003])
```

In [195]:

```
a = np.random.randn(5)
print(a)
b = np.random.randn(5)
print(b)
print(np.maximum(a,b))
```

```
[-0.08910137 -0.64009628  1.26357974 -1.56426029  1.45134406]
[-0.52614467 -0.41357558 -0.50932676  2.3490309  -0.49498766]
[-0.08910137 -0.41357558  1.26357974  2.3490309   1.45134406]
```

In [196]:

```
a = np.random.randn(7)*2
print(a)
```

```
[-0.05457446 -0.6625821  -4.28696535 -0.08918278  1.44384513 -2.61001565
  2.36430836]
```

In [197]:

```
rem,whole = np.modf(a)
print(rem)
print(whole)
```

```
[-0.05457446 -0.6625821  -0.28696535 -0.08918278  0.44384513 -0.61001565
  0.36430836]
[-0. -0. -4. -0.  1. -2.  2.]
```

In [202]:

```
a = np.random.randn(1,5)
print(a)
np.where(a>0,1,-1)
```

```
[[-1.37334471  0.0541442  -0.18929178 -1.84866613  1.25029512]]
```

Out[202]:

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

In [204]:

```
a = np.array([1,2,3])
b = np.array([4,5,6])
c = np.array([True, True, False])
res = [(x if z else y) for x,y,z in zip(a,b,c)]
print(res)
```

```
[1, 2, 6]
```

Mathematical and Statistical Methods

- sum
- mean
- std, var
- min, max
- argmin, argmax (indices of max and min elements)
- cumsum, cumprod

In [205]:
```
a = np.random.randn(2,3)
print(a)
```

```
[[-0.9642694  -0.14366507  0.09834248]
 [-0.2295116   0.72171773  0.34130615]]
```

In [206]:
```
a.mean()
```

Out[206]:
```
-0.02934661757250771
```

In [207]:
```
np.mean(a)
```

Out[207]:
```
-0.02934661757250771
```

In [208]:
```
a.sum()
```

Out[208]:
```
-0.17607970543504625
```

In [209]:
```
a.mean(axis=1)
```

Out[209]:
```
array([-0.33653066,  0.27783743])
```

In [210]:
```
a.sum(axis=0)
```

Out[210]:
```
array([-1.19378099,  0.57805265,  0.43964863])
```

In [211]:
```
a = np.array([1,2,3,4])
a.cumsum()
```

Out[211]:
```
array([ 1,  3,  6, 10], dtype=int32)
```

In [212]:
```
a.cumprod()
```

Out[212]:
```
array([ 1,  2,  6, 24], dtype=int32)
```

In [216]:
```
a = np.array([[1,2,3],[4,5,6]])
a.cumsum(axis=1)
```

Out[216]:
```
array([[ 1,  3,  6],
       [ 4,  9, 15]], dtype=int32)
```

In [218]:
```
a.cumprod(axis=0)
```

Out[218]:
```
array([[ 1,  2,  3],
       [ 4, 10, 18]], dtype=int32)
```

Lavanya | Value Laden

In [229]:
```
a = np.arange(5)
print(a)
print(a>1)
print((a>1).sum()) #count of True values
```

```
[0 1 2 3 4]
[False False  True  True  True]
3
```

In [232]:
```
# Boolean values
b = np.array([True, True, False, False, False, True])
print(b.any())
print(b.all())
```

```
True
False
```

In [234]:
```
# Sorting the array
a = np.random.randn(5)
print(a)
a.sort()
print(a)
```

```
[0.31481789 1.49594617 0.93454314 1.4096351  0.48386977]
[0.31481789 0.48386977 0.93454314 1.4096351  1.49594617]
```

Array set operations -

- unique(a)
- intersect1d(a,b)
- union1d(a,b)
- in1d(a,b)
- setdiff1d(a,b)
- setxor1d(a,b) : symmetric difference

In [240]:
```
# unique and set logic

a = np.array([1,3,4,5,3,2,5,4])
np.unique(a)

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

Out[240]:
```
array([1, 2, 3, 4, 5])
```

In [243]:
```
a = np.array([1,3,4,5,3,2,5,4])
b = np.array([5,6,7,3,8,9,0,12])
print(np.intersect1d(a,b))
```

```
[3 5]
```

In [244]:
```
a = np.array([1,3,4,5,3,2,5,4])
b = np.array([5,6,7,3,8,9,0,12])
print(np.union1d(a,b))
```

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

In [245]:
```
a = np.array([1,3,4,5,3,2,5,4])
b = np.array([5,6,7,3,8,9,0,12])
print(np.in1d(a,b))
```

```
[False  True False  True  True False  True False]
```

In [246]:
```
a = np.array([1,3,4,5,3,2,5,4])
b = np.array([5,6,7,3,8,9,0,12])
print(np.setdiff1d(a,b))
```

```
[1 2 4]
```

In [247]:
```
a = np.array([1,3,4,5,3,2,5,4])
b = np.array([5,6,7,3,8,9,0,12])
print(np.setxor1d(a,b))
```

```
[ 0  1  2  4  6  7  8  9 12]
```

Files with Numpy

In [248]:
```
a = np.arange(50)
np.save('demo',a) #demo.npy file created
```

In [249]:
```
np.load('demo.npy')
```

Out[249]:
```
array([ 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, 32, 33,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49])
```

Linear Algebra with Numpy -

- diag : return oof diagonal elements as 1D array
- dot : matrix multiplication
- trace : sum of sum of the diagonal elements
- det : compute the matrix determinant
- eig : eigenvalues and eigenvectors of a square matrix
- inv : inverse of the square matrix

- pinv : moore-penrose pseudo-inverse of a matrix
- qr : QR decomposition
- svd : singular value decomposition
- solve : solve linear system, Ax=b for x
- lstsq : least-squares solution to Ax=b

```
In [251]: from numpy.linalg import *
```

```
In [252]: a = np.random.randn(2,5)
          m = a.T.dot(a)
          m
```

```
Out[252]: array([[ 0.14996476,  0.08020177, -0.03502961,  0.39667467,  0.11211768],
                 [ 0.08020177,  0.52529866, -1.00835587, -1.79079487,  0.0129893 ],
                 [-0.03502961, -1.00835587,  2.03832013,  4.01622519,  0.07016996],
                 [ 0.39667467, -1.79079487,  4.01622519,  9.36539545,  0.4915898 ],
                 [ 0.11211768,  0.0129893 ,  0.07016996,  0.4915898 ,  0.0883958 ]])
```

```
In [253]: inv(m)
```

```
Out[253]: array([[ 5.72493947e+16, -4.96547666e+16,  2.79407814e+15,
                  -1.35192181e+16,  7.64924355e+15],
                 [ 1.99333849e+15,  4.92431970e+15, -1.25632119e+15,
                   2.13852601e+15, -1.41474321e+16],
                 [ 3.35281800e+14,  2.87803395e+15,  1.71090376e+15,
                  -1.15478847e+14, -1.56410704e+15],
                 [ 2.33493951e+15, -1.88872350e+15, -1.19285684e+15,
                   2.01490356e+14, -2.85763233e+15],
                 [-8.61570425e+16,  7.04755154e+16,  1.91632862e+15,
                   1.58041216e+16,  9.51047575e+15]])
```

```
In [255]: q,r = qr(m)
          q,r
```

```
Out[255]: (array([[-0.33527273, -0.33843477,  0.68391641, -0.17985936, -0.52245866],
                  [-0.17930523, -0.48083651, -0.58243699,  0.41151966, -0.4775611 ],
                  [ 0.07831488,  0.69415398,  0.15427593,  0.51288058, -0.47452026],
                  [-0.88683636,  0.3497279 , -0.2245876 , -0.17097462,  0.10742325],
                  [-0.25065889, -0.22382761,  0.34465176,  0.7113532 ,  0.51211631]]),
           array([[-4.47291845e-01,  1.38483860e+00, -3.22714452e+00,
                   -7.92615971e+00, -4.92540602e-01],
                  [ 0.00000000e+00, -1.60887837e+00,  3.30049761e+00,
                    6.68001846e+00,  1.56655756e-01],
                  [ 0.00000000e+00,  0.00000000e+00,  4.00832761e-16,
                    8.37374691e-16, -2.31182379e-18],
                  [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
                   -3.21162860e-16, -6.24713842e-18],
                  [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
                    0.00000000e+00,  1.99383486e-17]]))
```

Random number generation -

- seed
- permutation
- shuffle
- rand
- randint
- randn
- binomial
- normal
- beta
- chisquare
- gamma
- uniform

```
In [256]: a = np.random.normal(size=(3,3))
          a

Out[256]: array([[-0.68128445,  1.52819997,  1.79828787],
                 [-0.37967636,  0.88591296,  0.25298548],
                 [ 1.42897383,  0.43276472,  1.34086431]])
```

```
In [261]: a = np.random.permutation(10)
          a

Out[261]: array([3, 5, 2, 0, 8, 1, 9, 6, 7, 4])
```

```
In [263]: a = np.random.randint(2,20)
          a

Out[263]: 2
```

```
In [265]: a = np.random.uniform(1,4)
          a

Out[265]: 1.2524186107362902
```

Lavanya | Value Laden