# NUMPY PART - 1 BY VIRAT TIWARI

October 17, 2023

NUMPY - Numpy stands for NUMERICAL PYTHON , it is used for the data processing and We have lists in Python that act as arrays, however they are slow to process. NumPy aims to provide an array object that is up to 50 times faster than traditional Python lists. It may be used to conduct a wide range of array-based mathematical operations. It extends Python with advanced analytical structures that ensure fast computations with arrays and matrices, as well as a large library of high-level mathematical functions that work with these arrays and matrices. NumPy arrays, unlike lists, are kept in a single continuous location in memory, allowing programmes to access and manipulate them quickly.

[1]:
```
# By using the " import " function we can easily import the numpy package in␣
↪python

# as - stands for alias , we can give any name to the alias like here we give "␣
↪np "

import numpy as np
```

[2]:
```
# l is our list

l = [1,2,3,4,5,6]
```

[3]:
```
# By using the " .array( ) " we can convert any list into the array

# This is one dimentional array
np.array(l)
```

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

[4]:
```
# Note - array ( ) , asarray ( ) and asanyarray ( ) all these three functions␣
↪are same and they are used to convert any list into array
```

[5]:
```
# This is how we can store in " ar " variable

ar=np.array(l)
```

[6]:
```
type(ar)

# Note - ndarray is N-dimentional array
```

[6]:
```
numpy.ndarray
```

[7]:
```
# This is two dimentional array

c=np.array([[1,2],[3,4]])
```

[8]:
```
np.asarray(l)
```

[8]:
```
array([1, 2, 3, 4, 5, 6])
```

[9]:
```
# a is our list

a=[2,3,6]
```

[10]:
```
#Here we convert "a" list into the array by using "asarray" function

np.asarray(a)
```

[10]:
```
array([2, 3, 6])
```

[11]:
```
b=np.matrix(l)
```

[12]:
```
b
```

[12]:
```
matrix([[1, 2, 3, 4, 5, 6]])
```

[13]:
```
# Note - By default Matrix is 2-dimentional

# Note - Array is a superclass in Munpy and Matrix is a subset of Array
```

[14]:
```
np.asarray(l)
```

[14]:
```
array([1, 2, 3, 4, 5, 6])
```

[15]:
```
a=[1,5,6]
```

[16]:
```
np.asanyarray(a)
```

[16]:
```
array([1, 5, 6])
```

[17]:
```
np.asanyarray(b)
```

[17]:
```
matrix([[1, 2, 3, 4, 5, 6]])
```

[18]:
```
a=np.array(l)
```

[19]:
```
a
```

[19]:
```
array([1, 2, 3, 4, 5, 6])
```

[20]:
```
# This is called swallow copy function

c=a
```

[21]:
```
c
```

[21]:
```
array([1, 2, 3, 4, 5, 6])
```

[22]:
```
a
```

[22]:
```
array([1, 2, 3, 4, 5, 6])
```

[23]:
```
c[0]
```

[23]:
```
1
```

[24]:
```
c[0]=100
```

[25]:
```
c
```

[25]:
```
array([100,   2,   3,   4,   5,   6])
```

[26]:
```
a
```

[26]:
```
array([100,   2,   3,   4,   5,   6])
```

[27]:
```
# This is called deep copy function

d=np.copy(a)
```

[28]:
```
d
```

[28]:
```
array([100,   2,   3,   4,   5,   6])
```

[29]:
```
a
```

[29]:
```
array([100,   2,   3,   4,   5,   6])
```

[30]:
```
a[1]=500
```

[31]:
```
a
```

[31]:
```
array([100, 500,   3,   4,   5,   6])
```

[32]:
```
d
```

[32]:
```
array([100,   2,   3,   4,   5,   6])
```

[33]:
```
# Note - In the end of the day we generate array or deal with the array in numpy
```

[34]:
```
np.fromfunction(lambda i,j:i==j,(3,3))
```

[34]:
```
array([[ True, False, False],
       [False,  True, False],
       [False, False,  True]])
```

[35]:
```
np.fromfunction(lambda i,j:i*j,(3,3))
```

[35]:
```
array([[0., 0., 0.],
       [0., 1., 2.],
       [0., 2., 4.]])
```

[36]:
```
# This is how we made arry from iterables

iterable=(i*i for i in range (5))
```

[39]:
```
np.fromiter(iterable,float)
```

[39]:
```
array([ 0.,  1.,  4.,  9., 16.])
```

[41]:
```
np.fromstring("256 456",sep=" ")
```

[41]:
```
array([256., 456.])
```

[42]:
```
np.fromstring("6 5",sep=" ")
```

[42]:
```
array([6., 5.])
```

DATA TYPES IN NUMPY -

[44]:
```
l=[2,3,4,5,6]
```

[45]:
```
np.array(l)
```

[45]:
```
array([2, 3, 4, 5, 6])
```

[46]:
```
ar
```

[46]:
```
array([1, 2, 3, 4, 5, 6])
```

[47]:
```
# dim ( ) function is used for getting the dimention of array

ar.ndim
```

[47]:
```
1
```

[51]:
```
ar2=np.array([[2,3,6,5],[4,7,8,9]])
```

[52]:
```
ar2
```

[52]:
```
array([[2, 3, 6, 5],
       [4, 7, 8, 9]])
```

[53]:
```
ar2.ndim
```

[53]:
```
2
```

[54]:
```
ar2.size
```

[54]:
```
8
```

[55]:
```
# Note - 2,4 represent 2 rows and 4 columns

ar2.shape
```

[55]:
```
(2, 4)
```

[58]:
```
ar2.dtype
```

[58]:
```
dtype('int64')
```

[60]:
```
ar22=np.array([(52,4.5,22,6.4),(10,34,1.9,2)])
```

[61]:
```
ar22
```

[61]:
```
array([[52. ,  4.5, 22. ,  6.4],
       [10. , 34. ,  1.9,  2. ]])
```

[62]:
```
ar22.dtype
```

[62]:
```
dtype('float64')
```

[63]:
```
range(5)
```

[63]:
```
range(0, 5)
```

[64]:
```
list(range(5))
```

[64]:
```
[0, 1, 2, 3, 4]
```

[71]:
```
np.arange(2.3,5.6,.3)
```

[71]:
```
array([2.3, 2.6, 2.9, 3.2, 3.5, 3.8, 4.1, 4.4, 4.7, 5. , 5.3])
```

[72]:
```
list(np.arange(2.3,5.6,.3))
```

[72]:
```
[2.3,
 2.5999999999999996,
 2.8999999999999995,
 3.1999999999999993,
 3.499999999999999,
 3.799999999999999,
 4.099999999999999,
 4.399999999999999,
 4.699999999999998,
 4.999999999999998,
 5.299999999999998]
```

[73]:
```
np.linspace(1,5,10)
```

[73]:
```
array([1.        , 1.44444444, 1.88888889, 2.33333333, 2.77777778,
       3.22222222, 3.66666667, 4.11111111, 4.55555556, 5.        ])
```

[75]:
```
# 1 - Dimentional

np.zeros(5)
```

[75]:
```
array([0., 0., 0., 0., 0.])
```

[76]:
```
# 2 - Dimentional

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

[76]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

[77]:
```
# 3 - Dimentional

np.zeros((3,4,2))
```

[77]:
```
array([[[0., 0.],
        [0., 0.],
        [0., 0.],
        [0., 0.]],

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

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

       [[0., 0.],
        [0., 0.],
        [0., 0.],
        [0., 0.]]])
```

[79]:
```
# 4 - Dimentional

np.zeros((3,4,2,5))
```

[79]:
```
array([[[[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., 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., 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., 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., 0., 0., 0., 0.]]]])
```

[80]:
```
np.ones(4)
```

[80]:
```
array([1., 1., 1., 1.])
```

[84]:
```
on =np.ones((2,3))
```

[83]:
```
np.ones((2,3,2))
```

[83]:
```
array([[[1., 1.],
        [1., 1.],
        [1., 1.]],

       [[1., 1.],
        [1., 1.],
        [1., 1.]]])
```

[85]:
```
on + 5
```

[85]:
```
array([[6., 6., 6.],
       [6., 6., 6.]])
```

[86]:
```
on*4
```

[86]:
```
array([[4., 4., 4.],
       [4., 4., 4.]])
```

[87]:
```
np.empty((3,5))
```

[87]:
```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

[88]:
```
# eye ( ) function is used for makung identity matrix in which we the 1 in the␣
 ↪diogonal of matrix

np.eye(4)
```

[88]:
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

[89]:
```
np.linspace(2,4,20)
```

[89]:
```
array([2.        , 2.10526316, 2.21052632, 2.31578947, 2.42105263,
       2.52631579, 2.63157895, 2.73684211, 2.84210526, 2.94736842,
       3.05263158, 3.15789474, 3.26315789, 3.36842105, 3.47368421,
       3.57894737, 3.68421053, 3.78947368, 3.89473684, 4.        ])
```

[90]:
```
np.logspace(2,5,20)
```

[90]:
```
array([   100.        ,    143.84498883,    206.91380811,    297.63514416,
          428.13323987,    615.84821107,    885.86679041,   1274.2749857 ,
         1832.98071083,   2636.65089873,   3792.69019073,   5455.59478117,
         7847.59970351,  11288.37891685,  16237.76739189,  23357.2146909 ,
        33598.18286284,  48329.30238572,  69519.27961776, 100000.        ])
```

[91]:
```
np.logspace(2,5,20,base = 2)
```

[91]:
```
array([ 4.        ,  4.46263167,  4.97877036,  5.55460457,  6.19703857,
        6.91377515,  7.71340798,  8.60552469,  9.60082176, 10.71123281,
       11.95007169, 13.3321921 , 14.87416568, 16.59448071, 18.5137638 ,
       20.65502717, 23.0439446 , 25.70915925, 28.68262708, 32.        ])
```

[92]:
```
# MOST USED FUNCTIONS OF NUMPY FOR MACHINE LEARNING
```

[93]:
```
# STEP 1 - GENERATE THE RANDOM DATA

# This is how we generate the random data

# .random.randn ( ) function is used for genreating the random data

np.random.randn(3,4)
```

[93]:
```
array([[ 0.38511381,  1.34047131,  0.02756226,  0.23338492],
       [ 0.09171129, -0.2927657 , -1.15867637, -0.31957988],
       [-1.20810435, -0.16189658, -0.52516483,  0.42808922]])
```

[94]:
```
# STEP 2 - STORE THE DATA IN VARIABLE

arr=np.random.randn(3,4)
```

[95]:
```
# STEP 3 - IMPORT THE PANDAS FOR CONVERTING RANDOM DATA INTO THE DATA FRAME

import pandas as pd
```

[96]:
```
# STEP 4 - THIS IS HOW WE CONVERT RANDOM DATA INTO DATA FRAME

# .DATAFRAME ( ) FUNCTION IS USED FOR CONVERTION ANY DATASET INTO THE DATA FRAME

pd.DataFrame(arr)
```

[96]:
```
          0         1         2         3
0  1.322910  1.680960  0.975839 -1.517264
1 -0.289448  2.404372 -1.034109  1.431583
2  0.200658 -0.028301  1.397539 -0.050462
```

[97]:
```
# This is another way to generate the random data

# .random.rand ( ) function is used for generating the random data

np.random.rand(3,4)
```

[97]:
```
array([[0.19892687, 0.93422698, 0.16482709, 0.19032771],
       [0.36050143, 0.41450434, 0.2503462 , 0.57563516],
       [0.16423318, 0.47914752, 0.25057244, 0.16902285]])
```

[99]:
```
# .random.randint is used for generating random data by passing the range

# Here we we pass the 1-100 and the matrix that we want is 3X4 which means 3␣
 ↪rows and 4 columns

np.random.randint(1,110,(3,4))
```

[99]:
```
array([[ 91,  48, 101,  32],
       [ 28,   8,  12,  24],
       [ 83,  36,  48,  62]])
```

[117]:
```
np.random.randint(1,110,(300,400))
```

[117]:
```
array([[ 19,  47,  49, …,  76,  94,  73],
       [ 62,  43,  84, …,  78,  60, 100],
       [ 84,   5,  21, …,  75,  36,  38],
       …,
       [ 50,  36,  92, …, 103,  88,  35],
       [ 78,  99,  18, …,  40,  88, 106],
       [ 64,  87,  43, …, 108,  53,  19]])
```

[118]:
```
# Bu using pandas or its function " .DataFrame " we can easily store the data

pd.DataFrame(np.random.randint(1,110,(300,400)))
```

[118]:
```
     0    1    2    3    4    5    6    7    8    9    …  390  391  392  \
0     39   92  103  108   71   78    9   30   27   86  …   23   51   17
1     46   84   48   74   47  102   35   55  102   77  …   53   11   81
2     62   51   96   67   62   55   81    3   17    7  …   98  107   37
3    102   54  104   12   10    7    1   43    8   87  …  103    4   24
4     50  106   52  106   88   94   46   55   13   82  …   85   28   15
..   …   …   …   …   …   …   …   …   …   …   …   …   …   …
```

```
295   40   13   67   99  102   97   76  107  104   17  …   43   26  102
296   73   16  107   84   89   85   75   53  101   62  …   70   65    4
297    3   44    1   60   73   90   51   98    7   96  …   53   79   86
298   66   49   46   25   25    8   35   11   28   60  …  109   77    1
299   29   16   87   14   25   95   54   99   59   56  …   83   73  100

     393  394  395  396  397  398  399
0     43   39   36   83   69  100   70
1     25   59   30   32   48   82   69
2     87  102   66   78   91   52   85
3     40   42   93   22  103   59   26
4     32   29   95   36   77   14   67
..   …   …   …   …   …   …   …
295   99   90   35    1  107   24   26
296   43   22   62   54   51   73   36
297   55   20   46   90   40    1   68
298   82   49   21    1   26   47   38
299   89   95  103   67   53   80   70

[300 rows x 400 columns]
```

[119]:
```
a=pd.DataFrame(np.random.randint(1,110,(300,400)))
```

[122]:
```
a.to_csv("Random Data.csv")
```

[128]:
```
a.to_html("Random Data.html")
```

[140]:
```
# Most used function

arr1=arr.reshape(6,2)
```

[141]:
```
arr1
```

[141]:
```
array([[ 1.32290982,  1.68096031],
       [ 0.97583928, -1.51726418],
       [-0.28944771,  2.40437225],
       [-1.03410895,  1.43158319],
       [ 0.20065818, -0.02830124],
       [ 1.39753895, -0.05046171]])
```

[142]:
```
# Indexing

arr1[1]
```

[142]:
```
array([ 0.97583928, -1.51726418])
```

[143]:
```
# SLICING

arr1[1][1]
```

[143]:
```
-1.517264183314288
```

[144]:
```
arr1[2:5]
```

[144]:
```
array([[-0.28944771,  2.40437225],
       [-1.03410895,  1.43158319],
       [ 0.20065818, -0.02830124]])
```

[146]:
```
# Slicing of array

arr1[2:5,1]
```

[146]:
```
array([ 2.40437225,  1.43158319, -0.02830124])
```

[159]:
```
np.random.randint(1,100,(5,5))
```

[159]:
```
array([[75, 21, 37, 23, 85],
       [71, 77, 99, 13, 85],
       [90, 11, 28, 36, 67],
       [22, 18, 46, 99, 12],
       [32, 94,  8, 36, 24]])
```

[160]:
```
m=np.random.randint(1,100,(5,5))
```

[161]:
```
# Now we pull the data that more than 50
```

[162]:
```
m[m>50]
```

[162]:
```
array([84, 93, 98, 83, 92, 95, 53, 84, 95, 95, 73, 72])
```

[163]:
```
m
```

[163]:
```
array([[84, 34, 46, 44, 17],
       [93, 13,  3, 98, 25],
       [83, 34, 26, 92, 49],
       [95, 29, 53, 84, 25],
       [48, 95, 95, 73, 72]])
```

[168]:
```
# Slicing or indexing is done in a very simple manner

m[2:4,[1,2]]
```

[168]:
```
array([[34, 26],
       [29, 53]])
```

[169]:
```
# Change or manipulation of data
```

[170]:
```
m[0][0]=500
```

[171]:
```
m
```

[171]:
```
array([[500,  34,  46,  44,  17],
       [ 93,  13,   3,  98,  25],
       [ 83,  34,  26,  92,  49],
       [ 95,  29,  53,  84,  25],
       [ 48,  95,  95,  73,  72]])
```

[172]:
```
m[2][3]=2500
```

[173]:
```
m
```

[173]:
```
array([[ 500,   34,   46,   44,   17],
       [  93,   13,    3,   98,   25],
       [  83,   34,   26, 2500,   49],
       [  95,   29,   53,   84,   25],
       [  48,   95,   95,   73,   72]])
```

FUNCTIONS INSIDE A ARRAY -

[175]:
```
arr5=np.random.randint(1,3,(3,3))
arr6=np.random.randint(1,3,(3,3))
```

[176]:
```
arr5
```

[176]:
```
array([[1, 2, 2],
       [1, 2, 1],
       [1, 1, 1]])
```

[177]:
```
arr6
```

[177]:
```
array([[2, 2, 2],
       [2, 2, 2],
       [2, 2, 2]])
```

[178]:
```
arr5+arr6
```

[178]:
```
array([[3, 4, 4],
       [3, 4, 3],
       [3, 3, 3]])
```

[179]:
```
arr5*arr6
```

[179]:
```
array([[2, 4, 4],
       [2, 4, 2],
       [2, 2, 2]])
```

[180]:
```
arr5**arr6
```

[180]:
```
array([[1, 4, 4],
       [1, 4, 1],
       [1, 1, 1]])
```

[181]:
```
arr5-arr6
```

[181]:
```
array([[-1,  0,  0],
       [-1,  0, -1],
       [-1, -1, -1]])
```

[182]:
```
arr5/arr6
```

[182]:
```
array([[0.5, 1. , 1. ],
       [0.5, 1. , 0.5],
       [0.5, 0.5, 0.5]])
```

[183]:
```
arr5+100
```

[183]:
```
array([[101, 102, 102],
       [101, 102, 101],
       [101, 101, 101]])
```

[184]:
```
arr5**2
```

[184]:
```
array([[1, 4, 4],
       [1, 4, 1],
       [1, 1, 1]])
```

BROADCASTING IN NUMPY -

[186]:
```
arr=np.zeros((4,4))
```

[187]:
```
row=np.array([1,2,3,4])
```

[188]:
```
arr+row
```

[188]:
```
array([[1., 2., 3., 4.],
       [1., 2., 3., 4.],
       [1., 2., 3., 4.],
       [1., 2., 3., 4.]])
```

[189]:
```
# T stands for the transpose

row.T
```

[189]:
```
array([1, 2, 3, 4])
```

[192]:
```
col=np.array([[1,2,3,4]])
```

[193]:
```
# T stands for the transpose

col.T
```

[193]:
```
array([[1],
       [2],
       [3],
       [4]])
```

[195]:
```
arr8=np.random.randint(1,4,(3,4))
```

[196]:
```
# sqrt ( ) function is used for getting the square

np.sqrt(arr8)
```

[196]:
```
array([[1.73205081, 1.41421356, 1.41421356, 1.73205081],
       [1.41421356, 1.        , 1.41421356, 1.        ],
       [1.41421356, 1.        , 1.73205081, 1.73205081]])
```

[198]:
```
# exp ( ) function is used for getting the exponent of the matrix

np.exp(arr8)
```

[198]:
```
array([[20.08553692,  7.3890561 ,  7.3890561 , 20.08553692],
       [ 7.3890561 ,  2.71828183,  7.3890561 ,  2.71828183],
       [ 7.3890561 ,  2.71828183, 20.08553692, 20.08553692]])
```

[200]:
```
# log ( ) function is used for getting the logarithm values of matrix of array

np.log10(arr8)
```

[200]:
```
array([[0.47712125, 0.30103   , 0.30103   , 0.47712125],
       [0.30103   , 0.        , 0.30103   , 0.        ],
       [0.30103   , 0.        , 0.47712125, 0.47712125]])
```

THANK YOU SO MUCH !!

YOURS VIRAT TIWARI :)