NUMPY BASICS

In [1]:
```
import numpy as np
```

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

In [3]:
```
x
```

Out[3]:
```
[1, 2, 3, 4, 5, 6]
```

In [4]:
```
type(x)
```

Out[4]:
```
list
```

In [5]:
```
y=np.array([1,2,3,4,5,6])
```

In [6]:
```
type(y)
```

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

In [7]:
```
y=np.array((1,2,3,4,5,6))
```

In [8]:
```
type(y)
```

Out[8]:
```
numpy.ndarray
```

In [9]:
```
y
```

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

In [10]:
```
x=[1,2,'vaibhav',69]
x
```

Out[10]:
```
[1, 2, 'vaibhav', 69]
```

In [11]:
```
y=np.array(x)
y
```

Out[11]:
```
array(['1', '2', 'vaibhav', '69'], dtype='<U11')
```

In [12]:
```
y=np.linspace(start=0,stop=20,num=10)
```

In [13]:
```
y
```

Out[13]:
```
array([ 0.        ,  2.22222222,  4.44444444,  6.66666667,  8.88888889,
       11.11111111, 13.33333333, 15.55555556, 17.77777778, 20.        ])
```

In [14]:
```
y=np.linspace(start=0,stop=20,num=10,endpoint=True)
y
```

Out[14]:
```
array([ 0.        ,  2.22222222,  4.44444444,  6.66666667,  8.88888889,
       11.11111111, 13.33333333, 15.55555556, 17.77777778, 20.        ])
```

In [15]:
```
y=np.linspace(start=0,stop=20,num=10,endpoint=False)
y
```

Out[15]:
```
array([ 0.,  2.,  4.,  6.,  8., 10., 12., 14., 16., 18.])
```

In [16]:
```
y=np.linspace(start=0,stop=20,num=10,endpoint=True,retstep=True) #retstrp gives the st
y
```

Out[16]:
```
(array([ 0.        ,  2.22222222,  4.44444444,  6.66666667,  8.88888889,
        11.11111111, 13.33333333, 15.55555556, 17.77777778, 20.        ]),
 2.2222222222222223)
```

In [17]:
```
y=np.linspace(start=0,stop=20,num=10,endpoint=True,retstep=False)
y
```

Out[17]:
```
array([ 0.        ,  2.22222222,  4.44444444,  6.66666667,  8.88888889,
       11.11111111, 13.33333333, 15.55555556, 17.77777778, 20.        ])
```

In [18]:
```
d=np.arange(start=1,stop=10,step=2) #arange is to create array with conditions
d
```

Out[18]:
```
array([1, 3, 5, 7, 9])
```

In [19]:
```
# 3D Array
#it is giving a normal nested list
z=[[1,2,3],[4,5,6],[7,8,9]]
z
```

Out[19]:
```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

In [20]:
```
z=np.array([[1,2,3],[4,5,6],[7,8,9]]) #if we use np.array it will give 3D array
z
```

Out[20]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [21]:
```
# array of 1's
# give the parameter
np.ones((3,3)) #means 3x3 matrix of one's
```

Out[21]:
```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

In [22]:
```
# now with zero array
np.zeros((4,5)) # means 4x5 matrix array of zeroes
```

Out[22]:
```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

In [23]:
```
 np.eye(4) # for identity matrix , means diagonal is 1
```

Out[23]:
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

RESHAPING

In [24]:
```
x=np.array([0,1,2,3,4,5,6,7,8,9])
x
```

Out[24]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [25]:
```
x.reshape((5,2)) # change the shape of 1D matrix to given parameters
```

Out[25]:
```
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7],
       [8, 9]])
```

SLICING

In [26]:
```
a=np.arange(10) # it will create array
```

In [27]:
```
a
```

Out[27]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [28]:
```
s= slice(2,7,2)
# start , end , step
```

In [29]:
```
a[s]
```

Out[29]:
```
array([2, 4, 6])
```

In [30]:
```
s=slice(1,6,2)
a[s]
```

Out[30]:
```
array([1, 3, 5])
```

In [31]:
```
x[2:7]
```

Out[31]:
```
array([2, 3, 4, 5, 6])
```

In [32]:
```
x[-6:-2]
```

Out[32]:
```
array([4, 5, 6, 7])
```

random number generation

In [33]:
```
np.random.rand(5) #gives random numbers
```

Out[33]:
```
array([0.68110443, 0.45754181, 0.25494662, 0.84117337, 0.85918516])
```

In [34]:
```
np.random.rand(5,4)  #gives random numbers in 2D format if parameters are given
```

Out[34]:
```
array([[0.92209983, 0.02417045, 0.84276401, 0.10438225],
       [0.4603582 , 0.77223048, 0.15738239, 0.5180234 ],
       [0.23100003, 0.17290245, 0.39223883, 0.3322044 ],
       [0.98161385, 0.45875319, 0.7689995 , 0.38281201],
       [0.15366398, 0.87479568, 0.9881652 , 0.40004842]])
```

In [35]:
```
np.random.randn(5) #
```

Out[35]:
```
array([-0.94090659,  1.6673833 ,  0.75433731,  1.71187966,  0.6880217 ])
```

In [36]:
```
np.random.randn(5,5)
```

Out[36]:
```
array([[ 1.44507747, -0.71567392, -0.17889438,  1.36488669, -0.93266188],
       [ 1.02326807,  1.20557492, -1.09176504, -0.56862943,  0.67058764],
       [ 1.41221982, -0.04747547, -0.14038541, -0.45103114, -0.15964131],
       [-0.35405496,  0.34435532,  0.8367741 ,  1.15638997,  2.47918998],
       [-0.78549939,  1.26269476, -3.73535807,  1.37878059, -0.32052681]])
```

randint

In [37]:
```
np.random.randint(low=1,high=100)
```

Out[37]:
```
20
```

In [38]:
```
np.random.randint(low=1,high=100,size=20)
```

Out[38]:
```
array([81, 13, 59, 48, 76, 42, 62, 11, 89, 45, 91, 32, 60, 30, 35, 73, 40,
       47, 57,  7])
```

seed

In [39]:
```
np.random.seed(69) # seed is for picking same value of random data
#69 is the starting point
# 4 is the random 4 numbers from starting point
np.random.rand(4)
```

Out[39]:
```
array([0.29624916, 0.80906772, 0.35025253, 0.78940926])
```

In [40]:
```
np.random.seed(69)
np.random.rand(5)
```

Out[40]:
```
array([0.29624916, 0.80906772, 0.35025253, 0.78940926, 0.56134898])
```

Broadcasting

In [41]:
```
#broadcasting means the things we will do to the array it will be reflected to whole a
```

```
arr=np.arange(0,10,1)
arr
```

Out[41]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [42]:
```
arr1=arr/10 #here we are div the whole array by 10
arr1
```

Out[42]:
```
array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])
```

In [43]:
```
arr*5 # here we are multiplying it by 5
```

Out[43]:
```
array([ 0,  5, 10, 15, 20, 25, 30, 35, 40, 45])
```

In [44]:
```
slice_arr=arr[0:6]
slice_arr
```

Out[44]:
```
array([0, 1, 2, 3, 4, 5])
```

In [45]:
```
slice_arr[:]=555
slice_arr
```

Out[45]:
```
array([555, 555, 555, 555, 555, 555])
```

In [46]:
```
arr_copy = arr.copy()
```

In [47]:
```
arr_copy[:]=1000
arr_copy
```

Out[47]:
```
array([1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000])
```

In [48]:
```
arr
```

Out[48]:
```
array([555, 555, 555, 555, 555, 555,   6,   7,   8,   9])
```

Conditional Selection

In [49]:
```
arr=np.arange(0,10,1)
arr
```

Out[49]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [50]:
```
boo_arr=arr>4 # it is checking the bool value with the condtion provided if the condit
#will return True else False
boo_arr
```

Out[50]:
```
array([False, False, False, False, False,  True,  True,  True,  True,
        True])
```

In [51]:
```
arr[arr>4]
```

Out[51]:
```
array([5, 6, 7, 8, 9])
```

In [52]:
```
arr
```

Out[52]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [53]:
```
arr+arr # array + array it will return the point wise array
```

Out[53]:
```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
```

In [54]:
```
arr * arr
```

Out[54]:
```
array([ 0,  1,  4,  9, 16, 25, 36, 49, 64, 81])
```

In [55]:
```
arr/arr # warning becoz of the zero , that's why the first value is Nan (not a number)
```

```
C:\Users\studentadmin\AppData\Local\Temp\ipykernel_13328\3965243577.py:1: RuntimeWarn
ing: invalid value encountered in true_divide
  arr/arr # warning becoz of the zero , that's why the first value is Nan (not a numb
er)
```

Out[55]:
```
array([nan,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.])
```

In [56]:
```
arr**9 # ** is raise
```

Out[56]:
```
array([        0,         1,       512,     19683,    262144,   1953125,
        10077696,  40353607, 134217728, 387420489], dtype=int32)
```

Universal Array Functions

In [57]:
```
arr
```

Out[57]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [58]:
```
np.sqrt(arr) # sq root of each number
```

Out[58]:
```
array([0.        , 1.        , 1.41421356, 1.73205081, 2.        ,
       2.23606798, 2.44948974, 2.64575131, 2.82842712, 3.        ])
```

In [59]:
```
np.exp(arr)
```

Out[59]:
```
array([1.00000000e+00, 2.71828183e+00, 7.38905610e+00, 2.00855369e+01,
       5.45981500e+01, 1.48413159e+02, 4.03428793e+02, 1.09663316e+03,
       2.98095799e+03, 8.10308393e+03])
```

In [60]:
```
np.sin(arr)
```

Out[60]:
```
array([ 0.        ,  0.84147098,  0.90929743,  0.14112001, -0.7568025 ,
       -0.95892427, -0.2794155 ,  0.6569866 ,  0.98935825,  0.41211849])
```

In [61]:
```
size = 100
```

In [62]:
```
x1=range(size)
y1=range(size)
```

In [63]:
```
x1
```

Out[63]:
```
range(0, 100)
```

In [64]:
```
y1
```

Out[64]:
```
range(0, 100)
```

In [65]:
```
%%timeit # it will give the time to execute this statement
[x1[i]*y1[i] for i in range(size)]
```

```
16 µs ± 145 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
```

In [66]:
```
x2=np.arange(size)
y2=np.arange(size)
```

In [67]:
```
%%timeit
x2*y2
```

```
385 ns ± 6.81 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
```

Append

In [68]:
```
a=np.array([[1,2,3],[4,5,6],[7,8,9]])
a
```

Out[68]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [69]:
```
a_row=np.append(a,[[21,22,69]],axis=0) #axis = 0 means adding a row
a_row                                  # axis = 1 means adding a coloum
```

Out[69]:
```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [21, 22, 69]])
```

In [70]:
```
a_col=np.array([34,23,55]).reshape(3,1)
a_col
```

Out[70]:
```
array([[34],
       [23],
       [55]])
```

In [71]:
```
a_col=np.append(a,a_col,axis=1)
a_col
```

Out[71]:
```
array([[ 1,  2,  3, 34],
       [ 4,  5,  6, 23],
       [ 7,  8,  9, 55]])
```

In [ ]:
```

```