NUMPY AND PANDAS

- it is most commonly used libraries for data science
- it is also called as ArrayOriented computing as numpy allows the user to work in arrays is in python list
- it is mostly used for scientific programming and fast computing of lists

In [2]:
```
from numpy import *
a = array([1,2,3,4,5])
print(a)
```

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

In [16]:
```
b =array([[1,2],[3,4],[5,6]])
print(b)
```

```
[[1 2]
 [3 4]
 [5 6]]
```

In [12]:
```
print(type(a))
print(a.dtype)
```

```
<class 'numpy.ndarray'>
int32
```

In [13]:
```
c = array([1,2,3,4,5,1.23,2.4])
# all will be converted to float
print(c)
print(c.dtype)
```

```
[1.   2.   3.   4.   5.   1.23 2.4 ]
float64
```

In [17]:
```
#dimention of a
print(a.shape)
#dimention of b
print(b.shape)
#dimention of c
print(c.shape)
```

```
(5,)
(3, 2)
(7,)
```

In [18]:
```
b.reshape(2,3)
```

Out[18]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [19]:
```
c[[0,2,3]]
```

Out[19]:
```
array([1., 3., 4.])
```

In [21]:
```
b[1,1]
```

Out[21]:
```
4
```

In [23]:
```
c[-1:0:-1]
```

Out[23]:
```
array([2.4 , 1.23, 5.  , 4.  , 3.  , 2.  ])
```

In [24]:
```
c[1:3]
```

Out[24]:
```
array([2., 3.])
```

In [26]:
```
d = array([[[1,2],[3,4]],[[4,5],[6,7]]])
d
```

Out[26]:
```
array([[[1, 2],
        [3, 4]],

       [[4, 5],
        [6, 7]]])
```

In [27]:
```
d.shape
```

Out[27]:
```
(2, 2, 2)
```

In [28]:
```
c =array([[1,2,3],[4,5,6],[7,8,9]])
c
```

Out[28]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [29]:
```
c[0:3]
```

Out[29]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [30]:
```
c[1]
```

Out[30]:
```
array([4, 5, 6])
```

In [50]:
```
c.dtype
```

Out[50]:
```
dtype('int32')
```

In [31]:
```
c[0]
```

Out[31]:
```
array([1, 2, 3])
```

In [32]:
```
c[0,1]
```

Out[32]:
```
2
```

In [33]:
```
c[[0,1]]
```

Out[33]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [34]:
```
c[[0,2]]
```

Out[34]:
```
array([[1, 2, 3],
       [7, 8, 9]])
```

In [36]:
```
c[0:1]
```

Out[36]:
```
array([[1, 2, 3]])
```

In [39]:
```
c[0:1,0:2]
```

Out[39]:
```
array([[1, 2]])
```

In [49]:
```
e = array(array(list(map(lambda x: x*x,c))))
print(e)
```

```
[[ 1  4  9]
 [16 25 36]
 [49 64 81]]
```

In [54]:
```
e [1,2]
```

Out[54]:
```
36
```

In [61]:
```
# arrange() function impl
print(arange(100),"\n\n")           #with only upperlimit
print(arange(10,44),"\n\n")         # with upper and lower limit
print(arange(0,100,10),"\n\n")      # with lower limit and upperlimit with step size
```

```
[ 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 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
 96 97 98 99] 


[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] 


[ 0 10 20 30 40 50 60 70 80 90] 
```

In [64]:
```
print(a.strides)
```

```
(4,)
```

speed comparision : list against array

In [74]:
```
k= range(100)
%timeit [i**5 for i in k]
```

```
28.7 µs ± 264 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

In [75]:
```
print(k)
```

```
range(0, 100)
```

In [76]:
```
k= arange(100)
%timeit k**5
```

```
971 ns ± 3.2 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

In [77]:
```
print(k)
```

```
[ 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 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
 96 97 98 99]
```

In [78]:
```
exit()
```

In [4]:
```
from numpy import *
```

In [5]:
```
c= range(1000)
d= [i**2 for i in range(1000)]
%timeit list(map(lambda x,y:x*y,c,d))
```

```
96.4 µs ± 1.02 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

In [8]:
```
crr = arange(1000)
drr = crr**2
%timeit crr*drr
```

```
1.25 µs ± 6.87 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

```
- numpy for numpy arithematic operation the number of elements in each array should be the same
```

In [9]:
```
c =array([[1,2,3],[4,5,6],[7,8,9]])
c
```

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

In [10]:
```
zeros(4)
```

Out[10]:
```
array([0., 0., 0., 0.])
```

In [11]:
```
ones(4)
```

Out[11]:
```
array([1., 1., 1., 1.])
```

In [20]:
```
random.random((2,2))
```

Out[20]:
```
array([[0.90578031, 0.23530218],
       [0.75700342, 0.37047118]])
```

In [13]:
```
full((4),4)
```

Out[13]:
```
array([4, 4, 4, 4])
```

In [18]:
```
ones((3,3),dtype=int16)
```

Out[18]:
```
array([[1, 1, 1],
       [1, 1, 1],
       [1, 1, 1]], dtype=int16)
```

In [24]:
```
eye(4,dtype=int)
```

Out[24]:
```
array([[1, 0, 0, 0],
       [0, 1, 0, 0],
       [0, 0, 1, 0],
       [0, 0, 0, 1]])
```

In [27]:
```
diag([1,2,3,4,5])
```

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

In [34]:
```
c =array([[1,2,3],[4,5,6],[7,8,9]])
print(c,"\n\n")
print(c.T)
```

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


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

In [41]:
```
d = array([[1,2,1],[1,1,2],[2,1,1]])
linalg.inv(d)
```

Out[41]:
```
array([[-0.25, -0.25,  0.75],
       [ 0.75, -0.25, -0.25],
       [-0.25,  0.75, -0.25]])
```

In [46]:
```
random.random()
```

Out[46]:
```
0.634641357998184
```

In [58]:
```
50*random.random()+3
```

Out[58]:
```
33.55481823526476
```

In [84]:
```
qw=50*random.random((3,3))+3
qw
```

Out[84]:
```
array([[46.35684988,  8.7803952 , 22.35482008],
       [52.44253722, 11.36865947, 44.33039833],
       [ 6.41355878,  6.59844539, 46.69799243]])
```

In [82]:
```
random.randint(1,30)
```

Out[82]:
```
29
```

In [93]:
```
linspace(1,100,10)# lower limit # upper limit # no of equal slices required
```

Out[93]:
```
array([  1.,  12.,  23.,  34.,  45.,  56.,  67.,  78.,  89., 100.])
```

multi dimentional array

In [99]:
```
k=arange(24).reshape(2,3,4)
k
# no of elements = 2*3*4 = 24
# to perform lossless calculations on numpy the no of elements must be equal to the no
```

Out[99]:
```
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]]])
```

In [100]:
```
k.reshape(4,2,3) # accepted element [24]== used element [24]
```

Out[100]:
```
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]]])
```

In [102]:
```
k=arange(25).reshape(2,3,4) # cannot be reshaped due to in competability
# accepted element [25] != used element [24]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-102-eabab0ab280f> in <module>
----> 1 k=arange(25).reshape(2,3,4) # cannot be reshaped due to in competability
      2 # accepted element [25] != used element [24]

ValueError: cannot reshape array of size 25 into shape (2,3,4)
```

In [106]:
```
################################################################################
```

In [104]:
```
c =array([[1,2,3],[4,5,6],[7,8,9]])
c
```

Out[104]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [105]:
```
c[0]
```

Out[105]:
```
array([1, 2, 3])
```

In [112]:
```
c.T[0]
```

Out[112]:
```
array([1, 4, 7])
```

In [113]:
```
c[1]
```

Out[113]:
```
array([4, 5, 6])
```

In [114]:
```
c.T[1]
```

Out[114]:
```
array([2, 5, 8])
```

In [115]:
```
c[2]
```

Out[115]:
```
array([7, 8, 9])
```

In [116]:
```
c.T[2]
```

Out[116]:
```
array([3, 6, 9])
```

In [119]:
```
c>4 # boolean value for all the comparisions with 4 [all c>4]
```

Out[119]:
```
array([[False, False, False],
       [False,  True,  True],
       [ True,  True,  True]])
```

In [120]:
```
c<4  # boolean value for all the comparisions with 4 [all c<4]
```

Out[120]:
```
array([[ True,  True,  True],
       [False, False, False],
       [False, False, False]])
```

In [123]:
```
c[(c>4) & (c<7)]
```

Out[123]:
```
array([5, 6])
```

In [134]:
```
s1 = arange(10)
s1
```

Out[134]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [135]:
```
s2 = s1
s2[0]=99  # the identifiers in numpy module store the references to the memory so chang
s2
```

Out[135]:
```
array([99,  1,  2,  3,  4,  5,  6,  7,  8,  9])
```

In [127]:
```
s1
```

Out[127]:
```
array([99,  1,  2,  3,  4,  5,  6,  7,  8,  9])
```

In [129]:
```
s3 = s1.copy() # and so copy function of python  will cpoy the data to identifier not t
s3
```

Out[129]:
```
array([99,  1,  2,  3,  4,  5,  6,  7,  8,  9])
```

In [130]:
```
s3[0]=300
s3
```

Out[130]:
```
array([300,   1,   2,   3,   4,   5,   6,   7,   8,   9])
```

In [131]:
```
s1
```

Out[131]:
```
array([99,  1,  2,  3,  4,  5,  6,  7,  8,  9])
```

In [136]:
```
##########################################################################################
##########################################################################################
```

In [144]:
```
###################### horizontal and virtical merging ######################  stacki
```

In [138]:
```
a= array([[1,2,3],[4,5,6]])
b= array([[8,9,0],[6,7,2]])
a
```

Out[138]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [139]:
```
b
```

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

In [142]:
```
q = vstack((a,b))
q
```

Out[142]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [8, 9, 0],
       [6, 7, 2]])
```

In [143]:
```
p = hstack((a,b))
p
```

Out[143]:
```
array([[1, 2, 3, 8, 9, 0],
       [4, 5, 6, 6, 7, 2]])
```

```
#####################################  other functions for matrices
#########################################################
```

In [146]:
```
t = tan(a)
t
```

Out[146]:
```
array([[ 1.55740772, -2.18503986, -0.14254654],
       [ 1.15782128, -3.38051501, -0.29100619]])
```

In [148]:
```
c = cos(a)
c
```

Out[148]:
```
array([[ 0.54030231, -0.41614684, -0.9899925 ],
       [-0.65364362,  0.28366219,  0.96017029]])
```

In [149]:
```
s = cos(a)
s
```

Out[149]:
```
array([[ 0.54030231, -0.41614684, -0.9899925 ],
       [-0.65364362,  0.28366219,  0.96017029]])
```

In [150]:
```
sqrt(a)
```

Out[150]:
```
array([[1.        , 1.41421356, 1.73205081],
       [2.        , 2.23606798, 2.44948974]])
```

In [151]:
```
exp(a)
```

Out[151]:
```
array([[  2.71828183,   7.3890561 ,  20.08553692],
       [ 54.59815003, 148.4131591 , 403.42879349]])
```

In [152]:
```
std(a)
```

Out[152]:
```
1.707825127659933
```

In [153]:
```
pow(a,5)
```

Out[153]:
```
array([[   1,   32,  243],
       [1024, 3125, 7776]], dtype=int32)
```

In [204]:
```
########################################  importing data from file ###################
```

In [155]:
```
f= open("mat.txt","w")
f.write("qwertyu")
f.close()
```

In [165]:
```
f = open("mat.txt")
print(f.read())
f.close()
```

```
qwertyu

12 34 12
112 56 7
127 8 43
14 1 0
```

In [163]:
```
x, y, z = loadtxt('mat.txt',
                  skiprows=1,
                  unpack=True)
w = vstack((x.T,y.T,z.T))
w
```

Out[163]:
```
array([[ 12., 112., 127.,  14.],
       [ 34.,  56.,   8.,   1.],
       [ 12.,   7.,  43.,   0.]])
```

```
##########################################################################################
```

In [166]:
```
su = sum(a+b)
su
```

Out[166]:
```
53
```

In [168]:
```
matsum = a+b
matsum
```

Out[168]:
```
array([[ 9, 11,  3],
       [10, 12,  8]])
```

```
########################################  matrix multiplication
#################################################################
```

In [170]:
```
matmul = dot(a,b.T)
matmul
```

Out[170]:
```
array([[26, 26],
       [77, 71]])
```

In [171]:
```
matmul2 = dot(a.T,b)
matmul2
```

Out[171]:
```
array([[32, 37,  8],
       [46, 53, 10],
       [60, 69, 12]])
```

```
########################################  PARALLEL MATRIX MULTIPLY
##########################################################
```

In [174]:
```
pllMULmat =a*b
pllMULmat
```

Out[174]:
```
array([[ 8, 18,  0],
       [24, 35, 12]])
```

In [179]:
```
b%a
```

Out[179]:
```
array([[0, 1, 0],
       [2, 2, 2]], dtype=int32)
```

In [178]:
```
b/a
```

Out[178]:
```
array([[8.        , 4.5       , 0.        ],
       [1.5       , 1.4       , 0.33333333]])
```

In [180]:
```
a+b
```

Out[180]:
```
array([[ 9, 11,  3],
       [10, 12,  8]])
```

In [181]:
```
a**b
```

Out[181]:
```
array([[    1,   512,     1],
       [ 4096, 78125,    36]], dtype=int32)
```

In [183]:
```
b//a
```

Out[183]:
```
array([[8, 4, 0],
       [1, 1, 0]], dtype=int32)
```

In [194]:
```
a/b
```

```
C:\Users\STUDENT\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: RuntimeWarning:
divide by zero encountered in true_divide
  """Entry point for launching an IPython kernel.
```

Out[194]:
```
array([[0.125     , 0.22222222,        inf],
       [0.66666667, 0.71428571, 3.        ]])
```

In [190]:
```
a.max()
```

Out[190]:
```
6
```

In [192]:
```
a.min()
```

Out[192]:
```
1
```

```
##############################################  determinant and eigen values
##################################################
```

In [197]:
```
area = array([[1,2,1],[2,1,1],[1,1,2]])
linalg.det(area)
```

Out[197]:
```
-3.999999999999999
```

In [199]:
```
area
```

Out[199]:
```
array([[1, 2, 1],
       [2, 1, 1],
       [1, 1, 2]])
```

In [198]:
```
linalg.eig(area)
```

Out[198]:
```
(array([ 4., -1.,  1.]),
 array([[-5.77350269e-01, -7.07106781e-01, -4.08248290e-01],
        [-5.77350269e-01,  7.07106781e-01, -4.08248290e-01],
        [-5.77350269e-01,  1.02381169e-16,  8.16496581e-01]]))
```

```
.
```

In [ ]:
```

```