Numpy_Soltani

In [1]:

```
import numpy as np
print(np.__version__)
```

```
1.18.1
```

In [2]:

```
a = np.array([1,2,3,4,5,6])
print(a)
a[2]=10
print(a[2])
print(a.shape)#the shape of a.
print(a.dtype)#the type of a.
print(a.ndim) #the dimantion number of a.
print(a.size) #the total size of a.
```

```
[1 2 3 4 5 6]
10
(6,)
int32
1
6
```

In [3]:

```
b = np.array([7,8,9,10,11,12])
c = a * b
print(a)
print(b)
print(c)
```

```
[ 1  2 10  4  5  6]
[ 7  8  9 10 11 12]
[ 7 16 90 40 55 72]
```

diffrances betwen arrays and lists

In [4]:

```
l = [1,2,3]
a = np.array([1,2,3])

l = l + [4]# add var 4 to end of list.
a = a + [4]# addup 4 to all cells!

print(l)
print(a)
```

```
[1, 2, 3, 4]
[5 6 7]
```

in * ?

In [5]:

```
l = l * 2# another l will addup!
a = a * 2# all cells * 2!
print(l)
print(a)
```

```
[1, 2, 3, 4, 1, 2, 3, 4]
[10 12 14]
```

DOT

In [6]:

```
l1 = [1,2,3,4]
```

```
l2 = [5,6,7,8]
a1 = np.array(l1)
a2 = np.array(l2)
#lists
dot1=0
for i in range (len(l1)):
    dot1 += l1[i] * l2[i]
#np
dot2 = np.dot(a1,a2)
#or...
dot3 = a1 @ a2 # really cool!
print(dot1)
print(dot2)
print(dot3)
```

```
70
70
70
```

DIMANTIONS

In [7]:

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

print(a, '\n')
print(a.shape,'\n')

print(a[0], '\n')
print(a[0][1], '\n')
print(a[0,1], '\n')
print(a[:,1], '\n')

print(a.T, '\n') # transpose
print(np.linalg.inv(b), '\n')#invers -> should be squre
print(np.linalg.det(b), '\n')#determinan of a matrix! -> should be squre
print(np.diag(a), '\n') # diag for a matrix
```

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

(2, 3)

[1 2 3]

2

2

[2 5]

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

[[ 3.15251974e+15 -6.30503948e+15  3.15251974e+15]
 [-6.30503948e+15  1.26100790e+16 -6.30503948e+15]
 [ 3.15251974e+15 -6.30503948e+15  3.15251974e+15]]

-9.51619735392994e-16

[1 5]
```

bool & np

In [8]:

```
bool_ind = a>2
print(bool_ind,'\n')
print(a[bool_ind], '\n')
#_____________________________________
b = np.where(a>2, a, -1)
print(b)
#_____________________________________
```

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

```
 [ True  True  True]]

[3 4 5 6]

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

even numbers

In [9]:

```
a = np.array([2,4,5,8,0,4,6,67,97,66,534,33,4,2,5,66,7,7,7657,10])
print(a,'\n')
even = np.argwhere(a%2==0).flatten()#It will return the places of even numbers in a
print(a[even])
```

```
[   2    4    5    8    0    4    6   67   97   66  534   33    4    2
    5   66    7    7 7657   10]

[  2   4   8   0   4   6  66 534   4   2  66  10]
```

RESHAPE

In [10]:

```
a = np.arange(1,101) #Creat a(100,1) array with (1-101).
print(a)
print(a.shape)
b = a.reshape(10,10) #Reshape a to 10*10 array,
print(b)
```

```
[  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  99 100]
(100,)
[[  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  99 100]]
```

concatenate

In [11]:

```
a = np.array([[1,2,3,4], [5,6,7,8]])
b = np.array([[9,10,11,12]])

c = np.concatenate((a,b),axis=0)#Normal
d = np.concatenate((a,b),axis=None)#constantly after each other
b2 = np.array([[9,10]])
e = np.concatenate((a,b2.T),axis=1)#in columns!
print(c,'\n')
print(d,'\n')
print(e,'\n')
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]

[ 1  2  3  4  5  6  7  8  9 10 11 12]

[[ 1  2  3  4  9]
 [ 5  6  7  8 10]]
```

brodCasting

In [12]:

```
a = np.array([[1,2,3,4], [5,6,7,8],[1,2,3,4], [5,6,7,8],[1,2,3,4], [5,6,7,8]])
b = np.array([1,0,0,1])
c = a + b #[1,0,0,1] will add to all rows of a!
print(c)
```

```
[[2 2 3 5]
 [6 6 7 9]
 [2 2 3 5]
 [6 6 7 9]
 [2 2 3 5]
 [6 6 7 9]]
```

Functions & axis

In [13]:

```
print(a)

print('********************sum************************')
print(a.sum(),'\n')
print(a.sum(axis=None),'\n') # Total
print(a.sum(axis=0),'\n') # in columns
print(a.sum(axis=1),'\n')# in rows

print('********************mean************************')

print(a.mean(axis=None),'\n') # Total
print(a.mean(axis=0),'\n') # in columns
print(a.mean(axis=1),'\n')# in rows

print('************************var********************')

print(a.var(axis=None),'\n') # Total
print(a.var(axis=0),'\n') # in columns
print(a.var(axis=1),'\n')# in rows

print('***************************std******************')

print(a.std(axis=None),'\n') # Total
print(a.std(axis=0),'\n') # in columns
print(a.std(axis=1),'\n')# in rows

print('***************************max/min******************')

print(a.min(axis=None),'\n') # Total
print(a.min(axis=0),'\n') # in columns
print(a.min(axis=1),'\n')# in rows
```

```
[[1 2 3 4]
 [5 6 7 8]
 [1 2 3 4]
 [5 6 7 8]
 [1 2 3 4]
 [5 6 7 8]]
********************sum************************
108

108

[18 24 30 36]

[10 26 10 26 10 26]

********************mean************************
4.5

[3. 4. 5. 6.]

[2.5 6.5 2.5 6.5 2.5 6.5]

************************var********************
5.25

[4. 4. 4. 4.]
```

```
[1.25 1.25 1.25 1.25 1.25 1.25]

***************************std******************
2.29128784747792

[2. 2. 2. 2.]

[1.11803399 1.11803399 1.11803399 1.11803399 1.11803399 1.11803399]

***************************max/min******************
1

[1 2 3 4]

[1 5 1 5 1 5]
```

# Dtype

In [14]:

```python
x = np.array([1.0 , 2.0])
print(x.dtype,'\n')
y = np.array([1.0 , 2.0], dtype = np.int64)
print(y.dtype)
```

```
float64

int64
```

# copy

In [15]:

```python
a = np.array([1,2,3])
b = a
b[0]=10
print(b,' : b \n')
print(a,' : a \n')
```

```
[10  2  3]  : b

[10  2  3]  : a
```

In [16]:

```python
# so what should we do?
a = np.array([1,2,3])
b = a.copy()#:)
b[0]=10
print(b,' : b \n')
print(a,' : a \n')
```

```
[10  2  3]  : b

[1 2 3]  : a
```

# Generate Array

In [17]:

```python
a = np.zeros((2,3))#deffult is float64
b = np.ones((2,3))
c = np.full((2,3),6.0)
d = np.eye(3)
e = np.arange(20) # [0-20]
f = np.linspace(0,30,6)# 6 elements in [0-30]
g = np.random.random((3,2))# 0-1
h = np.random.randn(3,2)# mean ~=0 , var~=1
i = np.random.randint(3,10,size=(4,4))#4-9
```

```
j = np.random.choice([-8, -7, -2, 5], size=10)

print(a,' : a \n')
print(b,' : b \n')
print(c,' : c \n')
print(d,' : d \n')
print(e,' : e \n')
print(f,' : f \n')
print(g,' : g \n')
print(h,' : h \n')
print(i,' : i \n')
print(j,' : j \n')
```

```
[[0. 0. 0.]
 [0. 0. 0.]]  : a

[[1. 1. 1.]
 [1. 1. 1.]]  : b

[[6. 6. 6.]
 [6. 6. 6.]]  : c

[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]  : d

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

[ 0.  6. 12. 18. 24. 30.]  : f

[[0.27647358 0.47245624]
 [0.25750313 0.06563744]
 [0.86559837 0.61268518]]  : g

[[-0.30682477 -0.93129002]
 [ 1.48638509 -1.34720641]
 [ 1.03250096 -0.57076521]]  : h

[[9 3 8 7]
 [4 5 3 3]
 [8 8 8 5]
 [9 8 5 9]]  : i

[-7 -7 -7 -2  5  5  5 -2 -7 -2]  : j
```

# 2 eq -> 2UnKwon

In [18]:

```
A = np.array([[1, 1], [1.5, 4]])
l = np.array([2200, 5050])
x = np.linalg.solve(A, l)
print(x)
```

```
[1500.  700.]
```

$\color{red}{\text{it will be like this:

}}$ x + y = 2200 1.5x + 4y = 5050

. . .

x=1500, y=700

load txt / csv

In [19]:

```
import pandas as pd
data = pd.read_csv('data.csv')
#data = pd.read_csv('data.csv', sep = r"\s+", header = None)
```

In [20]:

```
print(data)
```

```
     1  1.1  1149  17.818  -14.218
0    1    1  1189   9.211  -87.813
1    1    1  1109   3.461   69.421
2    1    1  1249 -88.324   -6.420
3    1    1  1289 -90.178  -94.091
4    1    1  1209 -90.520   90.251
..  ..  ...   ...     ...      ...
182  2    7  2709  -1.715   88.049
183  2    7  2609  85.851   94.804
184  2    7  1648  81.011  106.914
185  2    7  1628  60.923   83.155
186  2    7  1738   9.044   76.523

[187 rows x 5 columns]
```

Processing math: 100%