dav-numpy-lab1

March 22, 2024

[5]:
```
import numpy as np
a=np.array([[1,2,3],[4,5,6]])
b=a.reshape(3,2)
print(b)
```

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

[6]:
```
import numpy as np
b=np.arange(24)
print(b)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]
```

[7]:
```
import numpy as np
a=np.array([1,2,3])
print(a)
```

```
[1 2 3]
```

[8]:
```
import numpy as np
a=np.array([[1,2],[3,4]])
print(a)
```

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

[9]:
```
import numpy as np
a=np.array([1,2,3,4,5],ndmin=2)
print(a)
```

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

[10]:
```
import numpy as np
a=np.array([1,2,3],dtype=complex)
print(a)
```

```
[1.+0.j 2.+0.j 3.+0.j]
```

[11]:
```
import numpy as np
a=np.array([[1,2],[3,4]])
print (a.shape)
```

```
(2, 2)
```

[12]:
```
import numpy as np
a=np.array([[1,2,3],[3,4,4]])
a.shape=(3,2)
print(a)
```

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

[14]:
```
b=a.reshape(2,4,3)
print(b)
#b is having 3 dim
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-14-eb57f0243da7> in <cell line: 1>()
----> 1 b=a.reshape(2,4,3)
      2 print(b)
      3 #b is having 3 dim

ValueError: cannot reshape array of size 6 into shape (2,4,3)
```

[15]:
```
import numpy as np
a=np.arange(24)
a.ndim
```

[15]: 1

[16]:
```
#dtype of array is int8(1byte)
x=np.array([1,2,3,4,5],dtype=np.int8)
print(x.itemsize)
```

```
1
```

[18]:
```
#dtype of array is float32(4byte)
x=np.array([1,2,3,4,5],dtype=np.float32)
print(x.itemsize)
```

```
4
```

[19]:
```
x=np.array([1,2,3,4,5])
print(x.flags)
```

```
  C_CONTIGUOUS : True
  F_CONTIGUOUS : True
  OWNDATA : True
  WRITEABLE : True
  ALIGNED : True
  WRITEBACKIFCOPY : False
```

[20]:
```
x=np.empty([3,2],dtype=int)
print(x)
```

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

[21]:
```
x=np.zeros([3,2],dtype=int)
print(x)
```

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

[22]:
```
import numpy as np
c=np.linspace(5,10,5)#start,end ,number of points
c
```

[22]:
```
array([ 5.  ,  6.25,  7.5 ,  8.75, 10.  ])
```

[23]:
```
d=np.ones((3,5))
d
```

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

[24]:
```
x=np.zeros((3,3))
x
```

[24]:
```
array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])
```

[25]:
```
import numpy as np
y=np.eye(4)
y
```

[25]:
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

[26]:
```
y=np.eye(3,2)
y
```

[26]:
```
array([[1., 0.],
       [0., 1.],
       [0., 0.]])
```

[27]:
```
a=np.diag([1,2,3,4])
a
```

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

[28]:
```
a=np.random.rand(4)
a
```

[28]:
```
array([0.43421795, 0.43009836, 0.66449872, 0.43549066])
```

[29]:
```
a=np.arange(10,dtype='float')
a
```

[29]:
```
array([0., 1., 2., 3., 4., 5., 6., 7., 8., 9.])
```

[30]:
```
b=np.array([1+2j,5+1j])
b.dtype
```

[30]:
```
dtype('complex128')
```

[31]:
```
c=np.array([True,False,True])
display(c.dtype)
```

```
dtype('bool')
```

[32]:
```
a=np.arange(10)
print(a)
print(a[5])
print(a[-1])
```

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

[33]:
```
b=np.diag([1,2,3])
print(b)
print(b[2,2])
```

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

[34]:
```
b[2,1]=10
b
```

[34]:
```
array([[ 1,  0,  0],
       [ 0,  2,  0],
       [ 0, 10,  3]])
```

[35]:
```
import numpy as np
a=np.arange(10)
print(a[1:10:2])#start_value:end_value(exclusive):step
```

```
[1 3 5 7 9]
```

[40]:
```
b=np.arange(10)
b[5:]=10#assign 10 from index 5 to end
print(b)
```

```
[ 0  1  2  3  4 10 10 10 10 10]
```

[41]:
```
a=np.arange(10)
b=a[::2]
np.shares_memory(a,b)
```

[41]:
```
True
```

[42]:
```
b[0]=10
print(b)
print(a)
```

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

[43]:
```
c=a[::2].copy()
np.shares_memory(a,c)
```

[43]:
```
False
```

[44]:
```
c[0]=5
print(c)
```

```
print(a)
```

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

[45]:
```
a=np.random.randint(0,20,15)
print(a)
```

```
[12 15  1  7  6  1 15  1  0  3 19 19 10  1 17]
```

[46]:
```
mask=(a%2==0)
```

[47]:
```
even_number=a[mask]
print(even_number)
```

```
[12  6  0 10]
```

[48]:
```
a[mask]=-1#it can be very useful to assign a new value to sub array
print(a)
```

```
[-1 15  1  7 -1  1 15  1 -1  3 19 19 -1  1 17]
```

[49]:
```
a=np.arange(0,100,10)
print(a)
```

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

[50]:
```
b=a[[2,3,5,2,4]]
print(b)
```

```
[20 30 50 20 40]
```

[51]:
```
a[[9,7]]=-200
print(a)
print(b)
```

```
[   0   10   20   30   40   50   60 -200   80 -200]
[20 30 50 20 40]
```

[52]:
```
a=np.arange(10)
print(a+1)
```

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

[53]:
```
print(a**2)
```

```
[ 0  1  4  9 16 25 36 49 64 81]
```

[54]:
```
b=np.ones(10)+1
print("b= ",b)
print("a-b=",a-b)
```

```
b=  [2. 2. 2. 2. 2. 2. 2. 2. 2. 2.]
a-b= [-2. -1.  0.  1.  2.  3.  4.  5.  6.  7.]
```

[55]:
```
c=np.diag([1,2,3,4])
print(c)
print("*"*100)
print(c*c)
print("*"*100)
print(c.dot(c))
```

```
[[1 0 0 0]
 [0 2 0 0]
 [0 0 3 0]
 [0 0 0 4]]
****************************************************************************************************
********************
[[ 1  0  0  0]
 [ 0  4  0  0]
 [ 0  0  9  0]
 [ 0  0  0 16]]
****************************************************************************************************
********************
[[ 1  0  0  0]
 [ 0  4  0  0]
 [ 0  0  9  0]
 [ 0  0  0 16]]
```

[56]:
```
import numpy as np#element comparision
a=np.array([1,2,5,4])
b=np.array([6,2,9,4])
print(a==b)
```

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

[57]:
```
print(a>b)
```

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

[58]:
```
print(np.array_equal(a,b))
```

```
False
```

[59]:
```
c=np.array([1,2,5,4])
print(np.array_equal(a,c))
```

```
True
```

logical operations

[60]:
```
a=np.array([1,0,0,1],dtype='bool')
b=np.array([0,1,0,1],dtype='bool')
```

[61]:
```
print(np.logical_or(a,b))
```

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

[64]:
```
print(np.logical_and(a,b))
```

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

[65]:
```
print(np.logical_not(a,b))
```

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

transcendental functions

[66]:
```
a=np.arange(5)+1
print(np.sin(a))
```

```
[ 0.84147098  0.90929743  0.14112001 -0.7568025  -0.95892427]
```

[67]:
```
print(np.log(a))
```

```
[0.         0.69314718 1.09861229 1.38629436 1.60943791]
```

[68]:
```
print(np.exp(a))
```

```
[  2.71828183   7.3890561   20.08553692  54.59815003 148.4131591 ]
```

shape mismatch

[69]:
```
a=np.array([1,2,3,4])
b=np.array([5,10])
print(a+b)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-69-f8e2422298ab> in <cell line: 3>()
      1 a=np.array([1,2,3,4])
      2 b=np.array([5,10])
----> 3 print(a+b)

ValueError: operands could not be broadcast together with shapes (4,) (2,)
```

basic reductions

[70]:
```
x=np.array([1,2,3,4])
print(np.sum(x))
```

```
10
```

[71]:
```
y=np.array([[1,2],[3,4]])
print(y)
print("*"*100)
print(y.T)
```

```
[[1 2]
 [3 4]]
****************************************************************************************************
[[1 3]
 [2 4]]
```

[72]:
```
print(y.sum(axis=0))#coloumn wise sum
```

```
[4 6]
```

[73]:
```
print(y.sum(axis=1))#row wise sum
```

```
[3 7]
```

[74]:
```
print(y.max())
```

```
4
```

[75]:
```
print(y.argmin())#index of min element
```

```
0
```

[76]:
```
print(y.argmax())#index of max element
```

```
3
```

logical reduction

[77]:
```
print(np.all([True,False,False]))#logical and
```

```
False
```

[78]:
```
print(np.any([True,False,False]))#logical or
```

```
True
```

[79]:
```
a=np.zeros((50,50))
print(np.any(a!=0))#cheaks wheather any element in a is not equal to zeros
```

```
False
```

statistics

[80]:
```
x=np.arange(1,10)
print(np.mean(x))
```

```
5.0
```

[81]:
```
print(np.median(x))
```

```
5.0
```

[91]:
```
y=np.array(([1,2,3],[4,5,6]))
print(np.mean(y,axis=0))#coloumn wise mean
print(np.mean(y,axis=1))
```

```
[2.5 3.5 4.5]
[2. 5.]
```

[83]:
```
print(np.std(x))
```

```
2.581988897471611
```

write a numpy program to convert a list of numeric value into a one-dimensional numpy array

[84]:
```
import numpy as np
a = [1, 2, 3, 4, 5]
x = np.array(a)
print(a)
print(x)
```

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

write a numpy program to create a 3*3 matrix with values ranging from 2,20

[85]:
```
import numpy as np
values = np.arange(2, 11)
matrix = values.reshape(3, 3)
print("3x3 Matrix with values ranging from 2 to 11:")
print(matrix)
```

```
3x3 Matrix with values ranging from 2 to 11:
[[ 2  3  4]
 [ 5  6  7]
 [ 8  9 10]]
```

[86]:
```
import numpy as np
arr = np.array([[3, 2, 1], [6, 5, 4]])
sorted_arr_first_axis = np.sort(arr, axis=0)
sorted_arr_last_axis = np.sort(arr, axis=-1)

print(arr)
print(sorted_arr_first_axis)
print(sorted_arr_last_axis)
```

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

numpy program to create a contiguous flATTENED ARRAY

[87]:
```
import numpy as np

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

b= a.flatten(order='F')

print(b)
```

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

numpy program to display all dates for the month of march 2017

[88]:
```
import datetime as dt

start_date = dt.date(2017, 3, 1)

end_date = dt.date(2017, 3, 31)

date_arr = np.arange(np.datetime64(start_date), np.datetime64(end_date) + np.
 ↪timedelta64(1, 'D'), dtype='datetime64[D]')

print(date_arr)
```

```
['2017-03-01' '2017-03-02' '2017-03-03' '2017-03-04' '2017-03-05'
 '2017-03-06' '2017-03-07' '2017-03-08' '2017-03-09' '2017-03-10'
 '2017-03-11' '2017-03-12' '2017-03-13' '2017-03-14' '2017-03-15'
 '2017-03-16' '2017-03-17' '2017-03-18' '2017-03-19' '2017-03-20'
 '2017-03-21' '2017-03-22' '2017-03-23' '2017-03-24' '2017-03-25'
 '2017-03-26' '2017-03-27' '2017-03-28' '2017-03-29' '2017-03-30'
 '2017-03-31']
```

[88]: