Numpy_VHA

January 6, 2025

1 Numpy

2 What is numpy?

- NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.
- At the core of the NumPy package, is the ndarray object. This encapsulates n-dimensional arrays of homogeneous data types

3 Numpy Arrays Vs Python Sequences

- NumPy arrays have a fixed size at creation, unlike Python lists (which can grow dynamically). Changing the size of an ndarray will create a new array and delete the original.
- The elements in a NumPy array are all required to be of the same data type, and thus will be the same size in memory.
- NumPy arrays facilitate advanced mathematical and other types of operations on large numbers of data. Typically, such operations are executed more efficiently and with less code than is possible using Python's built-in sequences.
- A growing plethora of scientific and mathematical Python-based packages are using NumPy arrays; though these typically support Python-sequence input, they convert such input to NumPy arrays prior to processing, and they often output NumPy arrays.

[2]:
```
import numpy as np
```

[1]:
```
%timeit sum(range(100000))
```

```
1.78 ms ± 49.2 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
```

[3]:
```
%timeit np.sum(np.arange(100000))
```

```
64.9 µs ± 336 ns per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

[7]:
```
# speed
# list
a = [i for i in range(10000000)]
b = [i for i in range(10000000,20000000)]

c = []
import time

start = time.time()
for i in range(len(a)):
  c.append(a[i] + b[i])
print(time.time()-start)
```

1.3717150688171387

[8]:
```
# numpy
import numpy as np
a = np.arange(10000000)
b = np.arange(10000000,20000000)

start = time.time()
c = a + b
print(time.time()-start)
```

0.06310343742370605

[9]:
```
# memory
a = [i for i in range(10000000)]
import sys

sys.getsizeof(a)
```

[9]: 89095160

[10]:
```
a = np.arange(10000000,dtype=np.int8)
sys.getsizeof(a)
```

[10]: 10000112

4 Creating Numpy Arrays

- np.array
- np.array with dtype
- np.arange
- with reshape
- np.ones and np.zeros
- np.random
- np.linspace

- np.identity

[4]:
```
# np.array
import numpy as np

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

```
[1 2 3]
<class 'numpy.ndarray'>
```

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

```
[1 2 3]
```

[11]:
```
# 2D and 3D
b = np.array([[1,2,3],[4,5,6]])
print(b)
print("*"*40)
c = np.array([[[1,2],[3,4]],[[5,6],[7,8]]])
print(c)
```

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

 [[5 6]
  [7 8]]]
```

[7]:
```
# dtype
np.array([1,2,3],dtype=float)
```

[7]:
```
array([1., 2., 3.])
```

[8]:
```
np.array([1,2,3],dtype=str)
```

[8]:
```
array(['1', '2', '3'], dtype='<U1')
```

[10]:
```
np.array([1,2,3],dtype=tuple)
```

[10]:
```
array([1, 2, 3], dtype=object)
```

[11]:
```
np.array([1,2,0],dtype=bool)
```

[11]:
```
array([ True,  True, False])
```

[12]:
```
np.array([1,2,0],dtype=complex)
```

[12]:
```
array([1.+0.j, 2.+0.j, 0.+0.j])
```

[13]:
```
# np.arange
np.arange(1,11,1)
```

[13]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

[14]:
```
np.arange(1,11,2)
```

[14]:
```
array([1, 3, 5, 7, 9])
```

[16]:
```
np.arange(11,1,-1)
```

[16]:
```
array([11, 10,  9,  8,  7,  6,  5,  4,  3,  2])
```

[17]:
```
# with reshape
np.arange(1,11).reshape(5,2)
```

[17]:
```
array([[ 1,  2],
       [ 3,  4],
       [ 5,  6],
       [ 7,  8],
       [ 9, 10]])
```

[18]:
```
np.arange(1,11).reshape(2,5)
```

[18]:
```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10]])
```

[19]:
```
np.arange(1,13).reshape(5,2)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-19-737dfc36139a> in <module>
----> 1 np.arange(1,13).reshape(5,2)

ValueError: cannot reshape array of size 12 into shape (5,2)
```

[20]:
```
np.arange(1,13).reshape(6,2)
```

[20]:
```
array([[ 1,  2],
       [ 3,  4],
       [ 5,  6],
       [ 7,  8],
       [ 9, 10],
```

```
       [11, 12]])
```

[21]:
```
np.arange(1,17).reshape(2,2,2,2)
```

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

        [[ 5,  6],
         [ 7,  8]]],


       [[[ 9, 10],
         [11, 12]],

        [[13, 14],
         [15, 16]]]])
```

[37]:
```
np.array([1,4,7,8,9,4,5,2,1,0,3,13]).reshape(6,2)
```

[37]:
```
array([[ 1,  4],
       [ 7,  8],
       [ 9,  4],
       [ 5,  2],
       [ 1,  0],
       [ 3, 13]])
```

[22]:
```
# np.ones and np.zeros
np.ones((3,4))
```

[22]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

[23]:
```
np.zeros((3,4))
```

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

[24]:
```
np.ones((3,4),dtype=int)
```

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

[25]:
```
# np.random
np.random.random((3,4))
```

[25]:
```
array([[7.66790583e-01, 2.78562999e-01, 2.17458415e-01, 5.15394925e-01],
       [1.46550894e-01, 2.49777909e-01, 5.39589866e-01, 1.27481946e-01],
       [4.82561439e-01, 2.14434094e-01, 5.39000369e-05, 8.57024103e-01]])
```

[28]:
```
np.random.randint(10,size=(5,2))
```

[28]:
```
array([[6, 5],
       [4, 4],
       [9, 9],
       [7, 9],
       [7, 1]])
```

[30]:
```
np.random.randint(20,size=(5,2,2))
```

[30]:
```
array([[[ 8,  7],
        [12, 13]],

       [[12, 12],
        [14, 11]],

       [[13, 12],
        [ 4, 18]],

       [[ 4, 17],
        [ 5, 14]],

       [[ 8,  3],
        [ 6, 16]]])
```

[32]:
```
# np.linspace
np.linspace(-10,10,10,dtype=int)
```

[32]:
```
array([-10,  -7,  -5,  -3,  -1,   1,   3,   5,   7,  10])
```

[33]:
```
np.linspace(-100,100,10,dtype=int)
```

[33]:
```
array([-100,  -77,  -55,  -33,  -11,   11,   33,   55,   77,  100])
```

[34]:
```
# np.identity
np.identity(3)
```

[34]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```


5 Array Attributes

- ndim

- shape
- size
- itemsize
- dtype

[39]:
```
a1 = np.arange(10,dtype=np.int32)
a2 = np.arange(12,dtype=float).reshape(3,4)
a3 = np.arange(8).reshape(2,2,2)
print(a1)
print()
print(a2)
print()
print(a3)
```

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

[[ 0.  1.  2.  3.]
 [ 4.  5.  6.  7.]
 [ 8.  9. 10. 11.]]

[[[0 1]
  [2 3]]

 [[4 5]
  [6 7]]]
```

[41]:
```
# ndim
print(a1.ndim)
print(a2.ndim)
print(a3.ndim)
```

```
1
2
3
```

[42]:
```
# shape
print(a1.shape)
print(a1)
print()
print(a2.shape)
print(a2)
print()
print(a3.shape)
print(a3)
```

```
(10,)
[0 1 2 3 4 5 6 7 8 9]

(3, 4)
```

```
[[ 0.  1.  2.  3.]
 [ 4.  5.  6.  7.]
 [ 8.  9. 10. 11.]]

(2, 2, 2)
[[[0 1]
  [2 3]]

 [[4 5]
  [6 7]]]
```

[43]:
```
# size
print(a1.size)
print(a1)
print()
print(a2.size)
print(a2)
print()
print(a3.size)
print(a3)
```

```
10
[0 1 2 3 4 5 6 7 8 9]

12
[[ 0.  1.  2.  3.]
 [ 4.  5.  6.  7.]
 [ 8.  9. 10. 11.]]

8
[[[0 1]
  [2 3]]

 [[4 5]
  [6 7]]]
```

[48]:
```
# itemsize
print(a1.itemsize)
print(a1)
print()
print(a2.itemsize)
print(a2)
print()
print(a3.itemsize)
print(a3)
```

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

```
8
[[ 0.  1.  2.  3.]
 [ 4.  5.  6.  7.]
 [ 8.  9. 10. 11.]]

4
[[[0 1]
  [2 3]]

 [[4 5]
  [6 7]]]
4
['1']
```

[49]:
```
# dtype
print(a1.dtype)
print(a2.dtype)
print(a3.dtype)
```

```
int32
float64
int32
```

6 Changing Datatype

[53]:
```
# astype
a3=a3.astype(np.int16)
```

[54]:
```
a3.dtype
```

[54]:
```
dtype('int16')
```

7 Array Operations

- reshape
- scalar operations
- relational
- vector operations

[59]:
```
a1 = np.arange(12).reshape(3,4)
a2 = np.arange(12,24).reshape(3,4)
print(a1)
print()
print(a2)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

```
[[12 13 14 15]
 [16 17 18 19]
 [20 21 22 23]]
```

[63]:
```
# scalar operations

# arithmetic
print("addition",a1+2)
print()
print("subtraction",a1-2)
print()
print("multipication",a1 * 2)
print()
print("divison",a1/2)
print()
print("power",a1 ** 2)
print()
print("f-divison",a1//2)
```

```
addition [[ 2  3  4  5]
 [ 6  7  8  9]
 [10 11 12 13]]

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

multipication [[ 0  2  4  6]
 [ 8 10 12 14]
 [16 18 20 22]]

divison [[0.  0.5 1.  1.5]
 [2.  2.5 3.  3.5]
 [4.  4.5 5.  5.5]]

power [[  0   1   4   9]
 [ 16  25  36  49]
 [ 64  81 100 121]]

f-divison [[0 0 1 1]
 [2 2 3 3]
 [4 4 5 5]]
```

[64]:
```
# relational
a2 == 15
```

[64]:
```
array([[False, False, False,  True],
       [False, False, False, False],
       [False, False, False, False]])
```

[65]:
```
a2>5
```

[65]:
```
array([[ True,  True,  True,  True],
       [ True,  True,  True,  True],
       [ True,  True,  True,  True]])
```

[66]:
```
a2>17
```

[66]:
```
array([[False, False, False, False],
       [False, False,  True,  True],
       [ True,  True,  True,  True]])
```

[67]:
```
a1>a2
```

[67]:
```
array([[False, False, False, False],
       [False, False, False, False],
       [False, False, False, False]])
```

[68]:
```
a1<a2
```

[68]:
```
array([[ True,  True,  True,  True],
       [ True,  True,  True,  True],
       [ True,  True,  True,  True]])
```

[69]:
```
# vector operations
# arithmetic
print("addition",a1+a2)
print()
print("subtraction",a1-a2)
print()
print("multipication",a1 * a2)
print()
print("divison",a1/a2)
print()
print("power",a1 ** a2)
print()
print("f-divison",a1//a2)
```

```
addition [[12 14 16 18]
 [20 22 24 26]
 [28 30 32 34]]

subtraction [[-12 -12 -12 -12]
 [-12 -12 -12 -12]
 [-12 -12 -12 -12]]
```

```
multipication [[  0  13  28  45]
 [ 64  85 108 133]
 [160 189 220 253]]

divison [[0.         0.07692308 0.14285714 0.2       ]
 [0.25       0.29411765 0.33333333 0.36842105]
 [0.4        0.42857143 0.45454545 0.47826087]]

power [[          0           1       16384    14348907]
 [          0 -1564725563  1159987200   442181591]
 [          0  1914644777 -1304428544  -122979837]]

f-divison [[0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]]
```

8 Array Functions

[89]:
```
a1 = np.random.random((3,3))
print(a1)
```

```
[[0.13943692 0.24849347 0.09540113]
 [0.11492873 0.50514922 0.57084673]
 [0.32226955 0.61744354 0.85962115]]
```

[90]:
```
a1 = np.round(a1*100)
print(a1)
```

```
[[14. 25. 10.]
 [11. 51. 57.]
 [32. 62. 86.]]
```

[91]:
```
# max
# 0 -> col and 1 -> row
np.max(a1,axis=0)
```

[91]: array([32., 62., 86.])

[93]:
```
np.max(a1)
```

[93]: 86.0

[92]:
```
np.max(a1,axis=1)
```

[92]: array([25., 57., 86.])

[94]:
```
#min
np.min(a1,axis=0)
```

[94]:
```
array([11., 25., 10.])
```

[95]:
```
np.min(a1,axis=1)
```

[95]:
```
array([10., 11., 32.])
```

[96]:
```
np.min(a1)
```

[96]:
```
10.0
```

[97]:
```
#sum
np.sum(a1,axis=0)
```

[97]:
```
array([ 57., 138., 153.])
```

[98]:
```
np.sum(a1,axis=1)
```

[98]:
```
array([ 49., 119., 180.])
```

[99]:
```
np.sum(a1)
```

[99]:
```
348.0
```

[100]:
```
np.prod(a1)
```

[100]:
```
19096152768000.0
```

[101]:
```
#prod
np.prod(a1,axis=0)
```

[101]:
```
array([ 4928., 79050., 49020.])
```

[79]:
```
np.prod(a1,axis=1)
```

[79]:
```
array([ 30912.,  10472., 116928.])
```

9 np.sort

- Return a sorted copy of an array.

[4]:
```
# code
import numpy as np
a = np.array([58, 69, 15, 43, 66, 72, 88, 44, 84, 68, 93, 77, 18, 89, 93])
a
```

[4]:
```
array([58, 69, 15, 43, 66, 72, 88, 44, 84, 68, 93, 77, 18, 89, 93])
```

[71]:
```
np.sort(a)[::-1]
```

[71]:
```
array([93, 93, 89, 88, 84, 77, 72, 69, 68, 66, 58, 44, 43, 18, 15])
```

[72]:
```
b = np.random.randint(1,100,24).reshape(6,4)
b
```

[72]:
```
array([[82, 39, 65, 97],
       [ 4, 48, 81, 39],
       [38, 26, 47, 94],
       [ 6, 42,  9, 45],
       [52, 75, 79, 94],
       [53, 68, 60, 67]])
```

[73]:
```
np.sort(b,axis=0)
```

[73]:
```
array([[ 4, 26,  9, 39],
       [ 6, 39, 47, 45],
       [38, 42, 60, 67],
       [52, 48, 65, 94],
       [53, 68, 79, 94],
       [82, 75, 81, 97]])
```

[74]:
```
np.sort(b,axis=1)
```

[74]:
```
array([[39, 65, 82, 97],
       [ 4, 39, 48, 81],
       [26, 38, 47, 94],
       [ 6,  9, 42, 45],
       [52, 75, 79, 94],
       [53, 60, 67, 68]])
```

[26]:
```
import numpy as np

arr = np.array([[39, 65, 82, 97],
                [ 4, 39, 48, 81],
                [26, 38, 47, 94],
                [ 6,  9, 42, 45],
                [52, 75, 79, 94],
                [53, 60, 67, 68]])
print(np.argsort(arr[:, 3]))
# Sorting by the 4th column (index 3), using np.sort on the axis 0
sorted_arr = arr[np.argsort(arr[:, 3])]

print(sorted_arr)
```

```
[3 5 1 2 4 0]
[[ 6  9 42 45]
 [53 60 67 68]
 [ 4 39 48 81]
 [26 38 47 94]
 [52 75 79 94]
 [39 65 82 97]]
```

10 np.concatenate

- numpy.concatenate() function concatenate a sequence of arrays along an existing axis.

[105]:
```
# code
c = np.arange(6).reshape(2,3)
d = np.arange(6,12).reshape(2,3)

print(c)
print(d)
```

```
[[0 1 2]
 [3 4 5]]
[[ 6  7  8]
 [ 9 10 11]]
```

[106]:
```
np.concatenate((c,d),axis=0)
```

[106]:
```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

[107]:
```
np.concatenate((c,d),axis=1)
```

[107]:
```
array([[ 0,  1,  2,  6,  7,  8],
       [ 3,  4,  5,  9, 10, 11]])
```

[108]:
```
e=d = np.arange(13,19).reshape(2,3)
```

[110]:
```
np.concatenate((c,d,e),axis=1)
```

[110]:
```
array([[ 0,  1,  2, 13, 14, 15, 13, 14, 15],
       [ 3,  4,  5, 16, 17, 18, 16, 17, 18]])
```

[111]:
```
np.concatenate((c,d,e),axis=0)
```

[111]:
```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [13, 14, 15],
```

```
       [16, 17, 18],
       [13, 14, 15],
       [16, 17, 18]])
```

[4]:
```
import numpy as np
a=np.arange(12).reshape(3,4)
b=np.arange(13,19).reshape(3,2)
np.concatenate((a,b),axis=0)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_18816\1903385360.py in <module>
      2 a=np.arange(12).reshape(3,4)
      3 b=np.arange(13,19).reshape(3,2)
----> 4 np.concatenate((a,b),axis=0)

<__array_function__ internals> in concatenate(*args, **kwargs)

ValueError: all the input array dimensions for the concatenation axis must matc
 ↪exactly, but along dimension 1, the array at index 0 has size 4 and the array
 ↪at index 1 has size 2
```

[5]:
```
import numpy as np
a=np.arange(12).reshape(3,4)
b=np.arange(13,19).reshape(3,2)
np.concatenate((a,b),axis=1)
```

[5]:
```
array([[ 0,  1,  2,  3, 13, 14],
       [ 4,  5,  6,  7, 15, 16],
       [ 8,  9, 10, 11, 17, 18]])
```

11 np.where

- The numpy.where() function returns the indices of elements in an input array where the given condition is satisfied.-

[120]:
```
print(a)
```

```
[58 69 15 43 66 72 88 44 84 68 93 77 18 89 93]
```

[121]:
```
# find all indices with value greater than 50
np.where(a>50)
```

[121]:
```
(array([ 0,  1,  4,  5,  6,  8,  9, 10, 11, 13, 14], dtype=int64),)
```

[122]:
```
# replace all values > 50 with 0#whwere(condition,true,false)
np.where(a>50,0,a)
```

[122]:
```
array([ 0,  0, 15, 43,  0,  0,  0, 44,  0,  0,  0,  0, 18,  0,  0])
```

[123]:
```
np.where(a%2 == 0,0,a)
```

[123]:
```
array([ 0, 69, 15, 43,  0,  0,  0,  0,  0,  0, 93, 77,  0, 89, 93])
```

[8]:
```
a=np.random.randint(1,100,24).reshape(6,4)
a
```

[8]:
```
array([[36, 61, 24, 43],
       [46, 83, 48, 64],
       [ 9, 69, 45, 49],
       [39, 76, 56, 77],
       [72, 93, 57, 15],
       [23, 29, 77, 37]])
```

[12]:
```
print(np.where(a%2==0))
```

```
(array([0, 0, 1, 1, 1, 3, 3, 4], dtype=int64), array([0, 2, 0, 2, 3, 1, 2, 0],
dtype=int64))
```

[14]:
```
print(np.where(a%2==0,a,0))
```

```
[[36  0 24  0]
 [46  0 48 64]
 [ 0  0  0  0]
 [ 0 76 56  0]
 [72  0  0  0]
 [ 0  0  0  0]]
```

12 Indexing and Slicing

[27]:
```
a1 = np.arange(10)
a2 = np.arange(12).reshape(3,4)
a3 = np.arange(8).reshape(2,2,2)
print(a1)
print()
print(a2)
print()
print(a3)
```

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

[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]

[[[0 1]
```

```
  [2 3]]

 [[4 5]
  [6 7]]]
```

[114]:
```
print(a1)
print(a1[-1])
```

[114]:
```
9
```

[116]:
```
print(a1)
print(a1[0])
```

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

[117]:
```
print(a1)
print(a1[-3])
```

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

[128]:
```
print(a2)
print(a2[1])
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
[4 5 6 7]
```

[28]:
```
print(a2)
print(a2[1][2])
print(a2[1,2])
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
6
6
```

[29]:
```
print(a2)
print(a2[-2][-2])
print(a2[-2,-2])
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
6
6
```

[130]:
```
print(a3)
print(",,,,,,,,")
print(a3[1])
```

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

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

[30]:
```
print(a3)
print(a3[1][0][1])
print(a3[1,0,1])
```

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

 [[4 5]
  [6 7]]]
5
5
```

[31]:
```
print(a3)
print(a3[-1][-2][-1])
print(a3[-1,-2,-1])
```

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

 [[4 5]
  [6 7]]]
5
5
```

[123]:
```
print(a1)
print(a1[2:5:1])
```

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

[124]:
```
print(a1)
print(a1[::-1])
```

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

[125]:
```
print(a1)
print(a1[-1:0:-2])
```

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

[127]:
```
print(a2)
print(".............")
print(a2[::-1])
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
.............
[[ 8  9 10 11]
 [ 4  5  6  7]
 [ 0  1  2  3]]
```

[131]:
```
print(a2)
print(".............")
print(a2[0,:])
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
.............
[0 1 2 3]
```

[132]:
```
print(a2)
print(".............")
print(a2[:,2])
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
.............
[ 2  6 10]
```

[133]:
```
print(a2)
print(".............")
print(a2[1:,1:3])
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
.............
[[ 5  6]
 [ 9 10]]
```

[134]:
```
print(a2)
print(".............")
print(a2[::2,::3])
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
…
[[ 0  3]
 [ 8 11]]
```

[136]:
```
print(a2)
print(".............")
print(a2[::2,1::2])
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
…
[[ 1  3]
 [ 9 11]]
```

[138]:
```
a3 = np.arange(1,28).reshape(3,3,3)
print(a3)
```

```
[[[ 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]]]
```

[140]:
```
print(a3)
print("*"*45)
print(a3[0,1,:])
```

```
[[[ 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]]]
**********
[4 5 6]
```

[143]:
```
#print(a3)
print("*"*45)
print(a3[1])
```

```
**********
[[10 11 12]
 [13 14 15]
 [16 17 18]]
```

[144]:
```
#print(a3)
print("*"*45)
print(a3[::2])
```

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

 [[19 20 21]
  [22 23 24]
  [25 26 27]]]
```

[142]:
```
#print(a3)
print("*"*45)
print(a3[:,::2,::2])
```

```
**********
[[[ 1  3]
  [ 7  9]]

 [[10 12]
  [16 18]]

 [[19 21]
  [25 27]]]
```

[145]:
```
#print(a3)
print("*"*45)
print(a3[1,:,1])
```

```
**********
[11 14 17]
```

[147]:
```
#print(a3)
print("*"*45)
print(a3[2,1:,1:])
```

```
*********************************************
[[23 24]
 [26 27]]
```

[148]:
```
#print(a3)
print("*"*45)
print(a3[::2,0,::2])
```

```
*********************************************
[[ 1  3]
 [19 21]]
```

[184]:
```
# Fancy Indexing
a = np.arange(24).reshape(6,4)
print(a)
a[:,[0,2,3]]
```

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

[184]:
```
array([[ 0,  2,  3],
       [ 4,  6,  7],
       [ 8, 10, 11],
       [12, 14, 15],
       [16, 18, 19],
       [20, 22, 23]])
```

[2]:
```
# Fancy Indexing
import numpy as np
a = np.arange(24).reshape(6,4)
print(a)
a[[0,2,3],:]
```

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

[2]:
```
array([[ 0,  1,  2,  3],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

[185]:
```
a = np.arange(24).reshape(6,4)
print(a)
a[1:5,[0,2,3]]
```

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

[185]:
```
array([[ 4,  6,  7],
       [ 8, 10, 11],
       [12, 14, 15],
       [16, 18, 19]])
```

[187]:
```
a = np.arange(24).reshape(6,4)
print(a)
a[[1,2,3],[0,2,3]]
```

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

[187]:
```
array([ 4, 10, 15])
```

[188]:
```
a = np.random.randint(1,100,24).reshape(6,4)
print(a)
```

```
[[60 88 27 43]
 [91 40 73 79]
 [78 81 54 35]
 [57 67 29 33]
 [24 48 73 21]
 [ 2 28 87 49]]
```

[16]:
```
 #Boolean Indexing
a = np.random.randint(1,100,24).reshape(6,4)
print(a)
```

```
[[43 52 42 38]
 [68 76 37 28]
 [99 79 64 96]
```

```
 [17 91 75 33]
 [34 27  9 44]
 [98 42 60 24]]
```

[17]:
```
# find all numbers greater than 50
print([a>50])
print(a[a > 50])
```

```
[array([[False,  True, False, False],
       [ True,  True, False, False],
       [ True,  True,  True,  True],
       [False,  True,  True, False],
       [False, False, False, False],
       [ True, False,  True, False]])]
[52 68 76 99 79 64 96 91 75 98 60]
```

[18]:
```
# find out even numbers
print([a % 2 == 0])
a[a % 2 == 0]
```

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

[18]:
```
array([52, 42, 38, 68, 76, 28, 64, 96, 34, 44, 98, 42, 60, 24])
```

[6]:
```
# find all numbers greater than 50 and are even

a[(a > 50) & (a % 2 == 0)]
```

[6]:
```
array([68, 90, 78, 56, 78, 94])
```

[7]:
```
# find all numbers not divisible by 7
a[~(a % 7 == 0)]
```

[7]:
```
array([ 8, 11, 69, 68,  3, 10, 50, 24, 59, 61,  6,  8, 90, 25, 78, 53, 78,
       71, 59, 94, 40, 16, 34])
```

13 Iterating

[149]:
```
print(a1)

for i in a1:
  print(i)
```

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

[153]:
```
print(a2)
print("*"*50)
for i in a2:
  print(i)
  for j in i:
    print(j)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
**************************************************
[0 1 2 3]
0
1
2
3
[4 5 6 7]
4
5
6
7
[ 8  9 10 11]
8
9
10
11
```

[157]:
```
print(a3)
print("*"*100)
for i in a3:
    print(i)
    print("*"*50)
    for j in i:
        print(j)
        for k in j:
            print(k)
```

```
[[[ 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]]]
**************************************************************************************
********************
[[1 2 3]
 [4 5 6]
 [7 8 9]]
************************************************
[1 2 3]
1
2
3
[4 5 6]
4
5
6
[7 8 9]
7
8
9
[[10 11 12]
 [13 14 15]
 [16 17 18]]
************************************************
[10 11 12]
10
11
12
[13 14 15]
13
14
15
[16 17 18]
16
17
18
[[19 20 21]
 [22 23 24]
 [25 26 27]]
```

```
**************************************************
[19 20 21]
19
20
21
[22 23 24]
22
23
24
[25 26 27]
25
26
27
```

[159]:
```
print(a3)
print(np.nditer(a3))
```

```
[[[ 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]]]
<numpy.nditer object at 0x000001F9C5A31E90>
```

[158]:
```
for i in np.nditer(a3):
  print(i)
```

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
```

- Reshaping
- Transpose
- Ravel

[162]:
```
print(a2)
print("*"*50)
print(a2.reshape(2,6))
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
**************************************************
[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]]
```

[163]:
```
print(a2)
print("*"*50)
print(np.transpose(a2))
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
**************************************************
[[ 0  4  8]
 [ 1  5  9]
 [ 2  6 10]
 [ 3  7 11]]
```

[165]:
```
print(a2.T)
```

```
[[ 0  4  8]
 [ 1  5  9]
 [ 2  6 10]
 [ 3  7 11]]
```

[166]:
```
print(a2)
print("*"*50)
print(a2.ravel())
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
**************************************************
[ 0  1  2  3  4  5  6  7  8  9 10 11]
```

[167]:
```
print(a3)
print("*"*50)
print(a3.ravel())
```

```
[[[ 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]]]
**************************************************
[ 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]
```

14 The array_split() function takes the following parameter values.

- array: This is the input array to be split. It is a required parameter.
- indices_or_sections: This is an integer representation of the number of the section of the array to be split. An error is raised if the number of splits specified is not possible. It is a required parameter.
- axis: This is the axis along which the split is done. It is an optional parameter.

Return value

- The array_split() function returns a list of sub-arrays of the input array.

[15]:
```
from numpy import array, array_split
# creating the input arrray
first_array = array([1, 2, 3, 4, 5, 6, 7, 8, 9])

# splitting the input array into 3 sub-arrays
my_array = array_split(first_array, 3)

# printing the split array
print(my_array)
```

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

[16]:
```
from numpy import array, array_split
# creating the input arrray
first_array = array([1, 2, 3, 4, 5, 6, 7, 8, 9])

# splitting the input array into 3 sub-arrays
my_array = array_split(first_array, 5)

# printing the split array
print(my_array)
```

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

[17]:
```
a=np.arange(12).reshape(3,4)
a
```

[17]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

[18]:
```
np.array_split(a,3,axis=1)
```

[18]:
```
[array([[0, 1],
        [4, 5],
        [8, 9]]),
 array([[ 2],
        [ 6],
        [10]]),
 array([[ 3],
        [ 7],
        [11]])]
```

[19]:
```
np.array_split(a,3,axis=0)
```

[19]:
```
[array([[0, 1, 2, 3]]), array([[4, 5, 6, 7]]), array([[ 8,  9, 10, 11]])]
```

[20]:
```
b=np.arange(12).reshape(3,2,2)
b
```

[20]:
```
array([[[ 0,  1],
        [ 2,  3]],

       [[ 4,  5],
        [ 6,  7]],

       [[ 8,  9],
        [10, 11]]])
```

[21]:
```
np.array_split(b,3,axis=0)
```

```
[21]: [array([[[0, 1],
               [2, 3]]]),
       array([[[4, 5],
               [6, 7]]]),
       array([[[ 8,  9],
               [10, 11]]])]
```

```
[22]: np.array_split(b,3,axis=1)
```

```
[22]: [array([[[0, 1]],

              [[4, 5]],

              [[8, 9]]]),
       array([[[ 2,  3]],

              [[ 6,  7]],

              [[10, 11]]]),
       array([], shape=(3, 0, 2), dtype=int32)]
```

```
[23]: np.array_split(b,3,axis=2)
```

```
[23]: [array([[[ 0],
               [ 2]],

              [[ 4],
               [ 6]],

              [[ 8],
               [10]]]),
       array([[[ 1],
               [ 3]],

              [[ 5],
               [ 7]],

              [[ 9],
               [11]]]),
       array([], shape=(3, 2, 0), dtype=int32)]
```

15 Broadcasting

- The term broadcasting describes how NumPy treats arrays with different shapes during arithmetic operations.
- The smaller array is "broadcast" across the larger array so that they have compatible shapes.

[20]:
```
# same shape
a = np.arange(6).reshape(2,3)
b = np.arange(6,12).reshape(2,3)

print(a)
print()
print(b)
print()
print(a+b)
```

```
[[0 1 2]
 [3 4 5]]

[[ 6  7  8]
 [ 9 10 11]]

[[ 6  8 10]
 [12 14 16]]
```

[10]:
```
print(a-b)
```

```
[[-6 -6 -6]
 [-6 -6 -6]]
```

[11]:
```
print(a*b)
```

```
[[ 0  7 16]
 [27 40 55]]
```

[21]:
```
print(a*b)
```

```
[[0.         0.14285714 0.25      ]
 [0.33333333 0.4        0.45454545]]
```

[22]:
```
print(a**b)
```

```
[[       0        1      256]
 [   19683  1048576 48828125]]
```

[23]:
```
print(a%b)
```

```
[[0 1 2]
 [3 4 5]]
```

[24]:
```
print(a//b)
```

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

[25]:
```
# diff shape
a = np.arange(6).reshape(2,3)
```

```
b = np.arange(3).reshape(1,3)

print(a)
print()
print(b)
print()
print(a+b)
```

```
[[0 1 2]
 [3 4 5]]

[[0 1 2]]

[[0 2 4]
 [3 5 7]]
```

16 matrix multiplication

[32]:
```
p = [[1, 2], [2, 3], [4, 5]]
q = [[4, 5, 1], [6, 7, 2]]
print("Matrix p :")
print(p)
print("Matrix q :")
print(q)

result = np.dot(p, q)

print("The matrix multiplication is :")
print(result)
```

```
Matrix p :
[[1, 2], [2, 3], [4, 5]]
Matrix q :
[[4, 5, 1], [6, 7, 2]]
The matrix multiplication is :
[[16 19  5]
 [26 31  8]
 [46 55 14]]
```

17 Broadcasting Rules

- 1. Make the two arrays have the same number of dimensions.

If the numbers of dimensions of the two arrays are different, add new dimensions with size 1 to the head of the array with the smaller dimension.

- 2. Make each dimension of the two arrays the same size.

If the sizes of each dimension of the two arrays do not match, dimensions with size 1 are stretched to the size of the other array. If there is a dimension whose size is not 1 in either of the two arrays, it cannot be broadcasted, and an error is raised.

[26]:
```
a = np.arange(12).reshape(4,3)
b = np.arange(3)

print(a)
print()
print(b)
print()
print(a+b)
```

```
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]

[0 1 2]

[[ 0  2  4]
 [ 3  5  7]
 [ 6  8 10]
 [ 9 11 13]]
```

[27]:
```
print(a*b)
```

```
[[ 0  1  4]
 [ 0  4 10]
 [ 0  7 16]
 [ 0 10 22]]
```

[28]:
```
a = np.arange(12).reshape(3,4)
b = np.arange(3)

print(a)
print()
print(b)
print()
print(a+b)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]

[0 1 2]
```

```
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_14796\2714742711.py in <module>
      6 print(b)
      7 print()
----> 8 print(a+b)

ValueError: operands could not be broadcast together with shapes (3,4) (3,)
```

[29]:
```
a = np.arange(3).reshape(1,3)
b = np.arange(3).reshape(3,1)

print(a)
print()
print(b)
print()
print(a+b)
```

```
[[0 1 2]]

[[0]
 [1]
 [2]]

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

[30]:
```
a = np.arange(3).reshape(1,3)
b = np.arange(4).reshape(4,1)

print(a)
print()
print(b)
print()
print(a + b)
```

```
[[0 1 2]]

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

[[0 1 2]
 [1 2 3]
 [2 3 4]
 [3 4 5]]
```

[31]:
```
a = np.array([1])
# shape -> (1,1)
b = np.arange(4).reshape(2,2)
# shape -> (2,2)

print(a)
print()
print(b)
print()
print(a+b)
```

```
[1]

[[0 1]
 [2 3]]

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

[32]:
```
a = np.arange(16).reshape(4,4)
b = np.arange(4).reshape(2,2)

print(a)
print(b)

print(a+b)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
[[0 1]
 [2 3]]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_14796\1506314148.py in <module>
      5 print(b)
      6
----> 7 print(a+b)

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

[ ]:
```

```