

Numpy Array

1. NumPy is a Python library used for working with arrays.
2. It also has functions for working in domain of linear algebra, fourier transform, and matrices.
3. NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.
4. NumPy is short for "Numerical Python".

Why We Use

1. We can use lists in place of array but they are slow and takes more time to process
2. Numpy array are faster than lists
3. The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.
4. Arrays are very frequently used in data science, where speed and resources are very important.

✓ Numpy Array Written In

NumPy is a Python library and is written partially in Python, but most of the parts that require fast computation are written in C or C++.

The source code for NumPy is located at this github repository <https://github.com/numpy/numpy>.

✓ Working with Numpy

Install

- pip install numpy

Import

- import numpy

Create Alias

- import numpy as np

✓ Check Time Which is Faster

```
#List
%timeit [i**1000 for i in range(10)]

🔄 21.6 µs ± 701 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)


#Numpy Array
import numpy as np
%timeit np.arange(10)**1000

🔄 2.08 µs ± 87.6 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)


def p1(n):
    l=[]
    for i in range(10):
        l.append(i**n)
    return l
%timeit p1(1000)

🔄 21.9 µs ± 1.15 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)


def p2(n):
    l=[2,1,3,4]
    return l
%timeit p2(1000)
```

 102 ns \pm 0.975 ns per loop (mean \pm std. dev. of 7 runs, 1000000 loops each)


```
def p3(n):  
    l=[2,'c','python',3.4,8]  
    return l  
%timeit p3(1000)
```

 107 ns \pm 2.03 ns per loop (mean \pm std. dev. of 7 runs, 1000000 loops each)


```
def p4(n):  
    l=['c++','c','python','java']  
    return l  
%timeit p4(1000)
```

 218 ns \pm 10 ns per loop (mean \pm std. dev. of 7 runs, 1000000 loops each)

```
%timeit [2,1,3,4]*1000
```


 7.75 μ s \pm 1.37 μ s per loop (mean \pm std. dev. of 7 runs, 100000 loops each)

```
import numpy as np  
%timeit np.array([2,1,3,4])*1000
```

 2.04 μ s \pm 146 ns per loop (mean \pm std. dev. of 7 runs, 100000 loops each)


✓ Create Our First Numpy Array

```
import numpy  
  
arr = numpy.array([2,1,4,5])  
print(arr)
```

 [2 1 4 5]


✓ Create List

```
l=[2,1,4,5]  
print(l)
```

 [2, 1, 4, 5]

✓ Create Alias and Use

```
import numpy as np  
  
arr = np.array([2,1,4,5])  
print(arr)
```

 [2 1 4 5]

✓ Check Version

```
np.__version__
```

✓ Check Type of Numpy Array

```
type(arr)
```

 numpy.ndarray

✓ Create Different Dimension Arrays

#Zero Dimension Array

```
import numpy as np
a = np.array(2)
print(a)
```

↔ 2

#One Dimension Array

```
import numpy as np
b = np.array([2,1,4])
print(b)
```

↔ [2 1 4]

#Two Dimension Array

```
import numpy as np
c = np.array([[1,2],[5,6]])
print(c)
```

↔
[[1 2]
[5 6]]

#Three Dimension Array

```
import numpy as np
d = np.array([[[1,2,3],[4,5,6]],[[4,2,1],[6,5,9]]])
print(d)
```

↔
[[[1 2 3]
[4 5 6]]

[[4 2 1]
[6 5 9]]]

#Four Dimension Array

```
import numpy as np
e = np.array([[[[1,2,3],[4,5,6]],[[4,2,1],[6,5,9]]],[[[1,2,3],[4,5,6]],[[4,2,1],[6,5,9]]]])
print(e)
```

↔
[[[[[1 2 3]
[4 5 6]]

[[4 2 1]
[6 5 9]]]

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

[[4 2 1]
[6 5 9]]]]]

▼ Check Dimensions

a.ndim

↔ 0

b.ndim

↔ 1

c.ndim

↔ 2

d.ndim

↔ 3

e.ndim

↔ 4

```
import numpy as np

a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

```
0
1
2
3
```

```
#Make Six Dimension Array
import numpy as np

arr = np.array([45,56], ndmin=6)

print(arr)
print('number of dimensions :', arr.ndim)
```

```
[[[[[[[45 56]]]]]]]
number of dimensions : 6
```

```
#Create a list of five elements
l = []
for i in range(5):
    x = int(input('enter element : '))
    l.append(x)
print(l)
```

```
enter element : 23
enter element : 56
enter element : 78
enter element : 12
enter element : 89
[23, 56, 78, 12, 89]
```

```
#Create an array of list
import numpy as np
print(np.array(l))
```

```
[23 56 78 12 89]
```

▼ Different Type of Arrays

1. 1 D Array - For Zeros and Ones: `np.zeros(columns)`
2. 2 D Array - For Zeros and Ones: `np.zeros((rows,columns))`
3. 3 D Array - For Zeros and Ones: `np.zeros((blocks,rows,columns))`

Note: Use the same pattern for N Dimension Array

▼ 1D Array

```
np.zeros(5)
```

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

▼ 2D Array

```
np.zeros((2,5))
```

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

```
np.zeros((3,2))
```

```
↔ array([[0., 0.],  
        [0., 0.],  
        [0., 0.]])
```

▼ 3D Array

```
np.zeros((2,4,3))
```

```
↔ array([[[0., 0., 0.],  
         [0., 0., 0.],  
         [0., 0., 0.],  
         [0., 0., 0.]],  
        [[0., 0., 0.],  
         [0., 0., 0.],  
         [0., 0., 0.],  
         [0., 0., 0.]])
```

```
np.ones(5)
```

```
↔ array([1., 1., 1., 1., 1.])
```

```
np.ones((2,5))
```

```
↔ array([[1., 1., 1., 1., 1.],  
        [1., 1., 1., 1., 1.]])
```

```
np.ones((3,2))
```

```
↔ array([[1., 1.],  
        [1., 1.],  
        [1., 1.]])
```

```
np.ones((2,4,3))
```

```
↔ array([[[1., 1., 1.],  
         [1., 1., 1.],  
         [1., 1., 1.],  
         [1., 1., 1.]],  
        [[1., 1., 1.],  
         [1., 1., 1.],  
         [1., 1., 1.],  
         [1., 1., 1.]])
```

▼ nD Array

```
np.zeros((2,4,3,5))
```

```
↔ array([[[[0., 0., 0., 0., 0.],  
          [0., 0., 0., 0., 0.],  
          [0., 0., 0., 0., 0.]],  
        [[0., 0., 0., 0., 0.],  
          [0., 0., 0., 0., 0.],  
          [0., 0., 0., 0., 0.]],  
        [[0., 0., 0., 0., 0.],  
          [0., 0., 0., 0., 0.],  
          [0., 0., 0., 0., 0.]],  
        [[0., 0., 0., 0., 0.],  
          [0., 0., 0., 0., 0.],  
          [0., 0., 0., 0., 0.]]],  
        [[[0., 0., 0., 0., 0.],  
          [0., 0., 0., 0., 0.],  
          [0., 0., 0., 0., 0.]],  
        [[0., 0., 0., 0., 0.],  
          [0., 0., 0., 0., 0.],  
          [0., 0., 0., 0., 0.]]])
```

```
[[0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0.],
 [0., 0., 0., 0., 0.]],

[[[0., 0., 0., 0., 0.],
   [0., 0., 0., 0., 0.],
   [0., 0., 0., 0., 0.]]])
```

```
np.ones((2,4,3,5))
```

[illegible]

- ▼ Range Array

```
import numpy as np
np.arange(10)
```

➡ array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

- Empty Array

```
arr = np.empty(4)
arr
```

```
→ array([4.64730676e-316, 0.00000000e+000, 6.42301491e+246, 1.07220128e+200])
```

```
arr = np.empty((4,3))
arr
```

```
→ array([[0.3, 1. , 1. ],
        [1. , 1. , 1. ],
        [1. , 1. , 1. ],
        [1. , 0.5, 1. ]])
```

```
arr = np.empty((4,3,2))
arr
```

```
⇒ array([[[-0.25, -0.5 ],
          [ 0.  , -0.25],
          [ 0.25, -0.5 ]],

         [[ 0.5 , -0.25],
          [ 0.25,  0.  ],
          [ 0.5 ,  0.25]]],
```

```
[[ 0.25,  0.5 ],
 [ 0.   ,  0.25],
 [-0.25,  0.5 ]],

[[-0.5 ,  0.25],
 [-0.25,  0.   ],
 [-0.5 , -0.25]]])
```

▼ Diagonal Array

▼ Unit Array

```
np.eye(4)
```

```
↔ array([[1., 0., 0., 0.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 0., 1.]])
```

```
np.eye(2)
```

```
↔ array([[1., 0.],
        [0., 1.]])
```

▼ Diagonal Array with diagonal value 1 for any array

```
np.eye(4,3)
```

```
↔ array([[1., 0., 0.],
        [0., 1., 0.],
        [0., 0., 1.],
        [0., 0., 0.]])
```

▼ Scaler Array

```
np.eye(4)*2
```

```
↔ array([[2., 0., 0., 0.],
        [0., 2., 0., 0.],
        [0., 0., 2., 0.],
        [0., 0., 0., 2.]])
```

```
np.eye(6)*5
```

```
↔ array([[5., 0., 0., 0., 0., 0.],
        [0., 5., 0., 0., 0., 0.],
        [0., 0., 5., 0., 0., 0.],
        [0., 0., 0., 5., 0., 0.],
        [0., 0., 0., 0., 5., 0.],
        [0., 0., 0., 0., 0., 5.]])
```

▼ Diagonal Array having any value on diagonal

```
np.diag([2,1,4,5])
```

```
↔ array([[2, 0, 0, 0],
        [0, 1, 0, 0],
        [0, 0, 4, 0],
        [0, 0, 0, 5]])
```

```
np.diag((2,7))
```

```
↔ array([[2, 0],
        [0, 7]])
```

▼ Requirement of elements in a range

```
np.linspace(5,20,num=5)
```

```
→ array([ 5. ,  8.75, 12.5 , 16.25, 20.  ])
```

```
np.linspace(1,2,num=6)
```

```
→ array([1. , 1.2, 1.4, 1.6, 1.8, 2. ])
```

✓ Array of Random Numbers

✓ rand():

function defined in the random module and generate random numbers between 0 and 1

```
np.random.rand(5)
```

```
→ array([0.78353799, 0.35884583, 0.7781049 , 0.53631117, 0.0480623 ])
```

```
np.random.rand(2)
```

```
→ array([0.72876549, 0.0213757 ])
```

```
np.random.rand(2,5)
```

```
→ array([[0.25509119, 0.02836408, 0.66851913, 0.43954464, 0.61377896],  
        [0.80932241, 0.65691926, 0.02561393, 0.75664088, 0.43993807]])
```

```
np.random.rand(4,3)
```

```
→ array([[0.45290732, 0.01048686, 0.72177979],  
        [0.45867937, 0.11581793, 0.42573422],  
        [0.03891808, 0.75602106, 0.71881213],  
        [0.58199685, 0.67852448, 0.04859443]])
```

```
np.random.rand(2,3,4)
```

```
→ array([[[0.97870652, 0.63394042, 0.20357622, 0.40777274],  
        [0.93031236, 0.48700301, 0.17537171, 0.91525033],  
        [0.78691878, 0.15725978, 0.13209009, 0.25560429]],  
        [[0.46974439, 0.55123979, 0.4414393 , 0.21367058],  
        [0.16929465, 0.40143534, 0.27216358, 0.59436984],  
        [0.95838275, 0.61196414, 0.02827833, 0.97535392]]])
```

✓ randn():

function defined in the random module and generate both positive negative values close to zero

```
np.random.randn(5)
```

```
→ array([-1.43807162,  0.03075063, -1.87372668,  0.84186615, -0.60564534])
```

```
np.random.randn(5,2)
```

```
→ array([[ 0.4604701 , -0.09062114],  
        [ 1.15499609, -0.02187291],  
        [-0.33198836, -1.05932534],  
        [-2.57421669,  0.18440957],  
        [ 0.77772461,  2.48276682]])
```

✓ randf()

function used to generate float values from 0 to 1 but 1 is excluded. This function is already defined in random module

```
np.random.randf(5)
```

```
→ array([0.83455828, 0.30537477, 0.80257385, 0.61140916, 0.62422214])
```



```
np.random.rand((5,3))
```

```
↔ array([[0.53061725, 0.63117972, 0.7988413 ],
        [0.51479241, 0.22930213, 0.55897052],
        [0.41294443, 0.56159002, 0.48164991],
        [0.63415499, 0.50495113, 0.43211107],
        [0.27391068, 0.6133115 , 0.23488454]])
```

```
np.random.rand((5,3,2))
```

```
↔ array([[[0.80630164, 0.73211639],
         [0.22788459, 0.25205377],
         [0.97699895, 0.6047011 ]],

        [[0.11525073, 0.19391645],
         [0.0893254 , 0.56887413],
         [0.80538081, 0.21248227]],

        [[0.95817743, 0.89354202],
         [0.78114798, 0.03682275],
         [0.23164338, 0.44349187]],

        [[0.04295169, 0.45659137],
         [0.15534465, 0.69145448],
         [0.27471862, 0.13690532]],

        [[0.97604934, 0.57625531],
         [0.0345954 , 0.61233601],
         [0.6625823 , 0.95718834]])])
```

✓ randint(): generate numbers between range

```
np.random.randint(1,20,5)
```

```
↔ array([ 7, 15, 15, 14, 19])
```

✓ Check Data Type

```
x = np.array([3,2,5])
x.dtype
```

```
↔ dtype('int64')
```

```
x = np.array([3.0,2.6,5.5])
x.dtype
```

```
↔ dtype('float64')
```

```
x = np.array(['p','y'])
x.dtype
```

```
↔ dtype('<U1')
```

```
x = np.array(['cpp','python'])
x.dtype
```

```
↔ dtype('<U6')
```

✓ Change Data Type

```
x = np.array([2,4.7], dtype='int8')
x.dtype
```

```
↔ dtype('int8')
```

```
x = np.array([2,4.7], dtype='i')
x.dtype
```

```
↔ dtype('int32')
```

```
x = np.array([2,4.7], dtype='float16')
x.dtype
```

```
dtype('float16')
```

```
x = np.array([2,4.7], dtype='f')
x.dtype
```

```
dtype('float32')
```

```
x = np.array([2,4.7])
np.int32(x).dtype
```

```
dtype('int32')
```

```
x = np.array([2,4.7])
np.int64(x).dtype
```

```
dtype('int64')
```

```
x = np.array([2,4.7])
np.float64(x).dtype
```

```
dtype('float64')
```

```
x = np.array([2,4,7])
y = x.astype(float)
y
```

```
array([2., 4., 7.])
```

✓ Arithmetic Operations

1. add(a1,a2)
2. subtract(a1,a2)
3. multiply(a1,a2)
4. divide(a1,a2)
5. mod(a1,a2)
6. power(a1,a2)
7. reciprocal(a1)

```
#Add with every element of array
x = np.array([2,4,6])
x+10
```

```
array([12, 14, 16])
```

```
#Subtract with every element of array
x = np.array([2,4,6])
x-2
```

```
array([0, 2, 4])
```

```
#Multiply with every element of array
x = np.array([2,4,6])
x*2
```


```
array([ 4,  8, 12])
```

```
#Divide with every element of array
x = np.array([2,4,6])
x/2
```

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


```
#Floor Divide with every element of array
x = np.array([2,4,6])
```

x//2

 array([1, 2, 3])


#Mod with every element of array

```
x = np.array([2,4,6])
x%2
```


 array([0, 0, 0])

#Add arrays


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

 array([3, 7, 11])

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


 array([3, 7, 11])

```
x = np.array([2,4,6])
y = np.array([1,3,5])
z = np.array([3,1,3])
x+y+z
```

 array([6, 8, 14])


#with add function, we can add two arrays

```
x = np.array([2,4,6])
y = np.array([1,3,5])
z = np.array([3,1,3])
np.add(x,y,z)
```

 array([3, 7, 11])


#with nesting, we can add

```
x = np.array([2,4,6])
y = np.array([1,3,5])
z = np.array([3,1,3])
np.add(np.add(x,y),z)
```

 array([6, 8, 14])


#Add arrays

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


 array([1, 1, 1])

#Add arrays

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

 array([1, 1, 1])

```
x = np.array([2,4,6])
y = np.array([1,3,5])
z = np.array([3,1,3])
x-y-z
```

 array([-2, 0, -2])

#only work with two arrays

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

```
z = np.array([3,1,3])
np.subtract(x,y,z)
```

```
↔ array([1, 1, 1])
```

```
x = np.array([2,4,6])
y = np.array([1,3,5])
z = np.array([3,1,3])
np.subtract(np.subtract(x,y),z)
```

```
↔ array([-2,  0, -2])
```

```
#Multiply arrays
x = np.array([2,4,6])
y = np.array([1,3,5])
x*y
```

```
↔ array([ 2, 12, 30])
```

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

```
↔ array([ 2, 12, 30])
```

```
#Divide arrays
x = np.array([2,4,6])
y = np.array([1,3,5])
x/y
```

```
↔ array([2.         , 1.33333333, 1.2         ])
```

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

```
↔ array([2.         , 1.33333333, 1.2         ])
```

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

```
↔ array([2, 1, 1])
```

```
#mod of arrays
x = np.array([2,4,6])
y = np.array([1,3,5])
np.mod(x,y)
```

```
↔ array([0, 1, 1])
```

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

```
↔ array([0, 1, 1])
```

```
#power of arrays
x = np.array([2,4,6])
y = np.array([1,3,5])
np.power(x,y)
```

```
↔ array([ 2,  64, 7776])
```

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

```
↔ array([ 2,  64, 7776])
```

```
#reciprocal of array
x = np.array([2,4,6])
np.reciprocal(x)
```

```
↩ array([0, 0, 0])
```

```
x = np.array([2,4,6])
1//x
```

```
↩ array([0, 0, 0])
```

```
x = np.array([[2,4,6],[1,6,5]])
print(np.add(x,y))
print(np.subtract(x,y))
print(np.multiply(x,y))
print(np.divide(x,y))
print(np.mod(x,y))
print(np.power(x,y))
print(np.reciprocal(x))
```

```
↩ [[ 3  7 11]
   [ 2  9 10]]
   [[1 1 1]
   [0 3 0]]
   [[ 2 12 30]
   [ 1 18 25]]
   [[2.          1.33333333 1.2          ]
   [1.          2.          1.          ]]
   [[0 1 1]
   [0 0 0]]
   [[ 2  64 7776]
   [ 1  216 3125]]
   [[0 0 0]
   [1 0 0]]
```

✓ Other Arithmetic Functions:

1. min()
2. max()
3. sqrt()
4. sin()
5. cos()
6. cumsum()

```
import numpy as np
x = np.array([12,4,26])
print(np.min(x))
print(np.max(x))
print(np.argmax(x)) #tell us the min value index
print(np.argmin(x)) #tell us the max value index
print(np.sqrt(x))
print(np.sin(x))
print(np.cos(x))
print(np.cumsum(x))
```

```
↩ 4
   26
   1
   2
   [3.46410162 2.          5.09901951]
   [-0.53657292 -0.7568025  0.76255845]
   [ 0.84385396 -0.65364362  0.64691932]
   [12 16 42]
```

```
x = np.array([[12,4,26],[3,6,5]])
print(np.min(x))
print(np.max(x))
print(np.argmax(x)) #tell us the min value index
print(np.argmin(x)) #tell us the max value index
print(np.sqrt(x))
print(np.sin(x))
print(np.cos(x))
print(np.cumsum(x))
```

```

↳ 3
   26
   3
   2
   [[3.46410162 2.          5.09901951]
    [1.73205081 2.44948974 2.23606798]]
   [[-0.53657292 -0.7568025   0.76255845]
    [ 0.14112001 -0.2794155  -0.95892427]]
   [[ 0.84385396 -0.65364362  0.64691932]
    [-0.9899925   0.96017029  0.28366219]]
   [12 16 42 45 51 56]

```

```

#column wise
x = np.array([[12,4,26],[3,6,5]])
print(np.min(x,axis=0))
print(np.max(x,axis=0))
print(np.argmin(x,axis=0)) #tell us the min value index
print(np.argmax(x,axis=0)) #tell us the max value index
print(np.sqrt(x))
print(np.cumsum(x,axis=0))

```

```

↳ [3 4 5]
   [12  6 26]
   [1 0 1]
   [0 1 0]
   [[3.46410162 2.          5.09901951]
    [1.73205081 2.44948974 2.23606798]]
   [[12  4 26]
    [15 10 31]]

```

```

#row wise
x = np.array([[12,4,26],[3,6,5]])
print(np.min(x,axis=1))
print(np.max(x,axis=1))
print(np.argmin(x,axis=1)) #tell us the min value index
print(np.argmax(x,axis=1)) #tell us the max value index
print(np.sqrt(x))
print(np.cumsum(x,axis=1))

```

```

↳ [4 3]
   [26  6]
   [1 0]
   [2 1]
   [[3.46410162 2.          5.09901951]
    [1.73205081 2.44948974 2.23606798]]
   [[12 16 42]
    [ 3  9 14]]

```

▼ Shape of Array

```

#1D Array
x = np.array([12,4,26])
x.shape

```

```

↳ (3,)

```

```

#2D Array
x = np.array([[12,4,26],[3,6,5]])
x.shape

```

```

↳ (2, 3)

```

```

#3D Array
x = np.array([[[12,4,26],[3,6,5]],[[12,4,26],[3,6,5]]])
x.shape

```

```

↳ (2, 2, 3)

```

```
#make 6 dimension array
x = np.array([12,4,26], ndmin=6)
print(x)
print(x.ndim)
print(x.shape)
```

```
[[[[[[[12  4 26]]]]]]]
6
(1, 1, 1, 1, 1, 3)
```

▼ Change the shape of Array

```
#one dimension to two dimension
x = np.array([1,2,3,4,5,6,7,8])
print(x)
print(x.shape)
print(x.ndim)
rx = x.reshape(4,2)
print(rx)
print(rx.shape)
print(rx.ndim)
```

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

```
#one dimension to three dimension
x = np.array([1,2,3,4,5,6,7,8])
print(x)
print(x.shape)
print(x.ndim)
rx = x.reshape(2,2,2)
print(rx)
print(rx.shape)
print(rx.ndim)
```

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

```
x = np.array([[1,2,3,4],[5,6,7,8]])
print(x)
print(x.shape)
print(x.ndim)
rx = x.reshape(-1)
print(rx)
print(rx.shape)
print(rx.ndim)
```

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

```
x = np.array([[1,2,3,4],[5,6,7,8]])
print(x)
print(x.shape)
print(x.ndim)
rx = x.reshape(4,2)
print(rx)
```

```
print(rx.shape)
print(rx.ndim)
```

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

```
x = np.array([[1,2,3,4],[5,6,7,8]])
print(x)
print(x.shape)
print(x.ndim)
rx = x.reshape(2,2,2)
print(rx)
print(rx.shape)
print(rx.ndim)
```

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

 [[5 6]
  [7 8]]]
(2, 2, 2)
3
```

▼ Indexing and Slicing of Array

▼ 1D Array

```
x = np.array([3,2,1,7,9,8])
print(x[2])
print(x[-2])
print(x[2:])
print(x[:2])
print(x[2:4])
print(x[-2:])
print(x[:-2])
print(x[-4:-2])
print(x[:])
print(x[:])
print(x[::1])
print(x[::2])
print(x[::3])
print(x[::4])
print(x[1::2])
print(x[:: -1])
print(x[-6:-2:1])
```

```
↔ 1
9
[1 7 9 8]
[3 2]
[1 7]
[9 8]
[3 2 1 7]
[1 7]
[3 2 1 7 9 8]
[3 2 1 7 9 8]
[3 2 1 7 9 8]
[3 1 9]
[3 7]
[3 9]
[2 7 8]
[8 9 7 1 2 3]
[3 2 1 7]
```



```
print(x[-6:-2:2])
print(x[-6:-1:3])
print(x[-2:-6:-1])
print(x[-2:-6:-2])
```

```
↔ [3 1]
   [3 7]
   [9 7 1 2]
   [9 1]
```

▼ 2D Array

```
x = np.array([[2,1,4,3],[5,1,4,7]])
print(x[1,2])
print(x[1,2:])
print(x[1,:2])
print(x[1,2:4])
print(x[1, 2])
print(x[-1, -2])
```

```
print(x[1, 2:])
print(x[-1, -2:])
```

```
print(x[1, :2])
print(x[-1, :-2])
```

```
print(x[1, 2:4])
print(x[-1, -2:])
```

```
↔ 4
   [4 7]
   [5 1]
   [4 7]
   4
   4
   [4 7]
   [4 7]
   [5 1]
   [5 1]
   [4 7]
   [4 7]
```

▼ 3D Array

```
x = np.array([[[2,1,4,3],[5,1,4,7]],[[2,1,4,3],[5,1,4,7]]])
print(x[1,1,2])
print(x[1,1,2:])
print(x[-1, -1, -2])
print(x[-1, -1, -2:])
```

```
↔ 4
   [4 7]
   4
   [4 7]
```

▼ Iteration on Array

```
#1D array
x = np.array([2,1,5,4])
for i in x:
    print(i)
```

```
↔ 2
   1
   5
   4
```

```
#2D array
x = np.array([[2,1,7,8],[3,2,9,8]])
for i in x:
    for j in i:
        print(j)
```

```
↔ 2
   1
   7
   8
   3
   2
   9
   8
```

```
#3D array
x = np.array([[2,1,4,3],[5,1,4,7]],[[2,1,4,3],[5,1,4,7]]])
for i in x:
    for j in i:
        for k in j:
            print(k)
```

```
↔ 2
   1
   4
   3
   5
   1
   4
   7
   2
   1
   4
   3
   5
   1
   4
   7
```

```
#Iterate using nditer() function over 3D Array
x = np.array([[2,1,4,3],[5,1,4,7]],[[2,1,4,3],[5,1,4,7]]])
for i in np.nditer(x):
    print(i)
```

```
↔ 2
   1
   4
   3
   5
   1
   4
   7
   2
   1
   4
   3
   5
   1
   4
   7
```

```
#Iterate using ndenumerate() function over 3D Array
x = np.array([[2,1,4,3],[5,1,4,7]],[[2,1,4,3],[5,1,4,7]]])
for i in np.ndenumerate(x):
    print(i)
```

```
↔ ((0, 0, 0), np.int64(2))
   ((0, 0, 1), np.int64(1))
   ((0, 0, 2), np.int64(4))
   ((0, 0, 3), np.int64(3))
   ((0, 1, 0), np.int64(5))
   ((0, 1, 1), np.int64(1))
   ((0, 1, 2), np.int64(4))
   ((0, 1, 3), np.int64(7))
   ((1, 0, 0), np.int64(2))
   ((1, 0, 1), np.int64(1))
   ((1, 0, 2), np.int64(4))
   ((1, 0, 3), np.int64(3))
   ((1, 1, 0), np.int64(5))
   ((1, 1, 1), np.int64(1))
```

```
((1, 1, 2), np.int64(4))
((1, 1, 3), np.int64(7))
```

```
x = np.array([[2,1,4,3],[5,1,4,7]],[[2,1,4,3],[5,1,4,7]])
for i,j in np.ndenumerate(x):
    print(i,j)
```

```
↔ (0, 0, 0) 2
   (0, 0, 1) 1
   (0, 0, 2) 4
   (0, 0, 3) 3
   (0, 1, 0) 5
   (0, 1, 1) 1
   (0, 1, 2) 4
   (0, 1, 3) 7
   (1, 0, 0) 2
   (1, 0, 1) 1
   (1, 0, 2) 4
   (1, 0, 3) 3
   (1, 1, 0) 5
   (1, 1, 1) 1
   (1, 1, 2) 4
   (1, 1, 3) 7
```

✓ Copy and View in array

1. copy does not affect the original data
2. view affect the original data

```
x = np.array([2,1,4,3])
y = x.copy()
y[0] = 10
x[-1]=50
print(x)
print(y)
```

```
↔ [ 2  1  4 50]
   [10  1  4  3]
```

```
x = np.array([2,1,4,3])
y = x.view()
x[-1]=50
y[0] = 10
print(x)
print(y)
```

```
↔ [10  1  4 50]
   [10  1  4 50]
```

✓ Join Array

```
x = np.array([2,1,4,3])
y = np.array([5,1,4,7])
z = np.concatenate((x,y))
print(z)
```

```
↔ [2 1 4 3 5 1 4 7]
```

```
x = np.array([[2,1],[4,3]])
y = np.array([[5,1],[4,7]])
z = np.concatenate((x,y))
print(z)
```

```
↔ [[2 1]
   [4 3]
   [5 1]
   [4 7]]
```

```
#concatenate along column wise
x = np.array([[2,1],[4,3]])
y = np.array([[5,1],[4,7]])
```

```
z = np.concatenate((x,y),axis=0)
print(z)
```

↔

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

```
#concatenate along row wise
x = np.array([[2,1],[4,3]])
y = np.array([[5,1],[4,7]])
z = np.concatenate((x,y),axis=1)
print(z)
```

↔

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

```
x = np.array([[2,1],[4,3]])
y = np.array([[5,1],[4,7]])
z = np.stack((x,y))
print(z)
```

↔

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

 [[5 1]
  [4 7]]]
```

```
x = np.array([[2,1],[4,3]])
y = np.array([[5,1],[4,7]])
z = np.hstack((x,y))
print(z)
```

↔

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

```
x = np.array([[2,1],[4,3]])
y = np.array([[5,1],[4,7]])
z = np.vstack((x,y))
print(z)
```

↔

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

› End of Code

[] ↳ 18 cells hidden