numpy

April 29, 2024

what is Array ?

`It a type of grid that contain values/data/information.`

What are the benifits of using numpy ?

```
It is faster than python list.
It consumes less memory.
Convenient to use.
Wide variety of mathematical functions
```

How to intall numpy ?

`open cmd and type following command " pip intall numpy ".`

Difference between Numpy Array and python list .

		LIST	Numpy
Data type storage	-	Store all type of data	Store only one type of data
Importing Module	-	No need to import Module	Need to import Module
Numerical operation	-	less efficiency	More efficient
Modification Cpabilities	-	Fast Modification	Less modification features
Memory	-	Consume more memory	Consume less memory
Speed	-	Slow	Fast
Convenient to use	-	No	Yes

Note :- use " %timeit" for checking the time taken by single line use " %timeit" for checking the time taken by whole program

```
    Eg(jupyter notebook only): %timeit[j**4 for j in range(1,9)]
        %timeit np.arange(1,9)**4             # will automaticall create array from 0 to 8 wit
```

`BUILT IN FUNCTIONS`

```
    Zeros :- All element will be 0 .
```

[2]:

```
import numpy as np
arr=np.zeros(5)
arr2=np.zeros((2,3))
print(arr)
print(arr2)
```

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

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

Ones :- All element will be 1 .

[24]:
```
arr=np.ones(5)
arr2=np.ones((2,3))
print(arr)
print(arr2)
```

```
[1. 1. 1. 1. 1.]

[[1. 1. 1.]
 [1. 1. 1.]]
```

Empty :- It will have previous memory data

[16]:
```
arr=np.empty(4)
print(arr)
```

```
[0.00000000e+000 1.09221928e-311 1.09221928e-311 1.09221928e-311]
```

Range

[23]:
```
arr=np.arange(4)
arr2=np.arange(2,4)
print(arr)
print(arr2)
```

```
[0 1 2 3]

[2 3]
```

Diagonal :- All diagonal element will be one.

[26]:
```
arr2=np.eye(3)
arr=np.eye(2,4)
print(arr)
print(arr2)
```

```
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]]

[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

LineSpace :- For defining equally spaced gap between elements

[29]:

```
arr=np.linspace(0,10,num=5)
print(arr)
```

```
[ 0.   2.5  5.   7.5 10. ]
```

full() : For making same valued array

[31]:

```
var=np.full((2,3),6)
print(var)
```

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

CREATING RANDOM ARRAYS

rand() :- Generate random values between 0-1 # (0 , 1)

[35]:

```
var=np.random.rand(4)
var2=np.random.rand(2,3)   # Here (2,3) is the dimension of an array as separate parameter
print(var)
print(var2)
```

```
[0.87034789 0.30370235 0.61844383 0.04726694]

[[0.91582055 0.26255411 0.77814962]
 [0.35261111 0.71902024 0.54348551]]
```

randn() :- Generate random values close to 0 .

[33]:

```
var=np.random.randn(4)
var2=np.random.randn(2,4)
print(var)
print(var2)
```

```
[ 2.91669517 -0.87388667 -1.22221261  0.48645842]

[[ 0.37346108  0.62974874 -0.13373476 -0.68745698]
 [ 0.32136739  1.2134603   1.11229282  0.81402857]]
```

ranf() :- Generate value in interval [0.0 , 1.0) # 1.0 not included .

[56]:

```
var=np.random.ranf(2)
var2=np.random.ranf((2,4))
print(var)
print(var2)
```

```
[0.32932786 0.51538185]

[[0.38977172 0.06802466 0.63871611 0.69798739]
 [0.26735088 0.92595842 0.35929806 0.45235006]]
```

```
randint() :- Generate value between user defined range ( min , max , total ).
```

[57]:

```
var=np.random.randint(2,4,5)
var2=np.random.randint(1,6,15)
print(var)
print(var2)
```

```
[3 3 3 3 3]

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

```
DATATYPES : bool_6 ( True or False )
        int_ (c long int32 or int64 )
        intc (c int int32 or int64 )
        int8 (-128 to 127)
        int16 (-32768 to 32767)
        uint8 ( 0 to 255 )
        uint16 ( 0 to 65535 )
        float_ ( float64 double precision)
        complex_ ( shorthand for complex128),etc
```

[58]:

```
var=np.array([1,2,3334,4])
print("Data Type",var.dtype)
```

```
Data Type int32
```

[59]:

```
var=np.array([1,2.0,3334,4])
print("Data Type",var.dtype)
```

```
Data Type float64
```

[60]:

```
var=np.array(["i","s","h","u"])
print("Data Type",var.dtype)
```

```
Data Type <U1
```

[62]:

```
var=np.array(["i","s","h","ua"])
print("Data Type",var.dtype)
```

```
Data Type <U2
```

[61]:

```
var=np.array([2,"i","s","h","u",1,2])
print("Data Type",var.dtype)
```

```
Data Type <U11
```

[69]:

```
var=np.array(["i","s","h","ua",3])
print("Data Type",var.dtype)
```

```
Data Type <U11
```

[70]:

```python
var=np.array([2,3,4],dtype=np.int8)
print("Data Type",var.dtype)
```

```
Data Type int8
```

[72]:

```python
var=np.array([2,3,4])     # Data Type int32
var2=np.int8(var)
print("Data Type",var2.dtype)
```

```
Data Type int8
```

astype() : Also used for defining the datatype

[73]:

```python
var=np.array([2,3,4])
var2=var.astype(float)
print(var)       # [2 3 4]
print(var2)
```

```
[2 3 4]
[2. 3. 4.]
```

ARITHMATIC OPERATIONS

[75]:

```python
a=np.array([1,2,3,4])
print("a+5 =",a+5)
print("a-5 =",a-5)
print("a/5 =",a/5)
print("a%5 =",a%5)
print("a//5 =",a/5)
```

```
a+5 = [6 7 8 9]
a-5 = [-4 -3 -2 -1]
a/5 = [0.2 0.4 0.6 0.8]
a%5 = [1 2 3 4]
a//5 = [0.2 0.4 0.6 0.8]
```

ARITHMATIC FUNCTIONS

    add(),subtract(),multipy(),divide(),floor_divide(),mod(),power()

[88]:

```python
a=np.array([1,2,3,4])
b=np.array([5,6,7,8])
print(f'{np.add(a,b) = }')
print(f'{np.subtract(a,b) = }')
print(f'{np.multiply(a,b) = }')
print(f'{np.divide(a,b) = }')
print(f'{np.floor_divide(a,b) = }')
print(f'{np.mod(a,b) = }')
print(f'{np.power(a,b) = }')
print(f'{np.sqrt(a) = }')
```

```
# we also have trignometry functions like sin(),cos(),etc
# we can also use multidimensional arrays but remember data passed in arrays␣
 ↪must be of same length
```

```
np.add(a,b) = array([ 6,  8, 10, 12])

np.subtract(a,b) = array([-4, -4, -4, -4])

np.multiply(a,b) = array([ 5, 12, 21, 32])

np.divide(a,b) = array([0.2       , 0.33333333, 0.42857143, 0.5       ])

np.floor_divide(a,b) = array([0, 0, 0, 0])

np.mod(a,b) = array([1, 2, 3, 4])

np.power(a,b) = array([    1,    64,  2187, 65536])

np.sqrt(a) = array([1.        , 1.41421356, 1.73205081, 2.        ])
```

`reciprocal()`

[83]:

```
a=np.array([21,2,3,4])
b=np.array([1,2,3,4])
c=np.array([5,6,7,8])
print(np.reciprocal(a))
print(np.reciprocal(b,c))
```

```
[0 0 0 0]

[1 0 0 0]
```

`tranpose()`

[85]:

```
var=np.array([1,2,3,4])        # [1 2 3 4]
var2=np.array([[1,2,3],[5,6,7]])
print(np.transpose(var))
print(np.transpose(var2))
```

```
[1 2 3 4]

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

`flatten()`

[87]:
```
var=np.array([1,2,3,4])
var2=np.array([[1,2,3],[5,6,7]])
print(var.flatten())
print(var2.flatten())
```

```
[1 2 3 4]

[1 2 3 5 6 7]
```

```
    cumsum()
```

[94]:
```
var=np.array([1,2,3])
var2=np.array([2,2,3,4])
print(np.cumsum(var))   # [ 1 1+2 1+2+3 ]
print(np.cumsum(var2))  # [ 2 2+2 2+2+3 2+2+3+4 ]
```

```
[1 3 6]
[ 2  4  7 11]
```

```
    min(),max()
```

[98]:
```
var=np.array([1,2,3,1,0,6,8,3,4])
print( "min ", np.min(var) )
print( "max ", np.max(var) )
```

```
min  0
max  8
```

[100]:
```
var=np.array([[1,2,3,1],[6,8,3,4]])
print( "min ", np.min(var) )
print( "max ", np.max(var) )
```

```
min  1
max  8
```

```
axis = 0
    According to Columns
axis = 1
    According to Rows
```

[105]:
```
var=np.array([[11,2,10,3],[5,8,3,14],[9,15,6,17]])
print( "min ", np.min(var , axis=0) )
print( "max ", np.max(var , axis=0) )
```

```
min  [5 2 3 3]
max  [11 15 10 17]
```

[106]:
```
var=np.array([[1,2,12,11],[6,8,3,4]])
print( "min ", np.min(var , axis=1) )
print( "max ", np.max(var , axis=1) )
```

```
min  [1 3]
max  [12  8]
```

argmin()/argmax() : Give position of min / max terms.

[108]:

```
var=np.array([[1,2,3,1],[3,5,1,6]])
print(np.argmin(var,axis=1))
print(np.argmax(var,axis=1))
print(np.argmin(var,axis=0))
print(np.argmax(var,axis=0))
```

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

ndim

[54]:

```
a = np.array([["agrawal","ishu",24]])
print(a.ndim)
```

```
2
```

shape

[8]:

```
a=np.array([1,2,3,4])
b=np.array([[1,2,3,4],[1,2,3,4]])
print(a,a.shape)
print(b,b.shape)
```

```
[1 2 3 4] (4,)

[[1 2 3 4]
 [1 2 3 4]] (2, 4)
```

ndmin

[9]:

```
a=np.array([1,2,3,4],ndmin=3)
print(a)                # [[[1 2 3 4]]]
print(a.shape)          # (1, 1, 4)   #(number of rows in each dimensional array)
print(a.ndim)           # 3
```

```
[[[1 2 3 4]]]
(1, 1, 4)
3
```

reshape

[15]:

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

```
print(x)
print(x.ndim,x.shape)
```

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

[21]:

```
b=np.array([1,2,3,4,5,6,7,8,9,10,11,12])  # shape (12,)
c=b.reshape(2,3,2)
print(c)
d=c.reshape(12)
h=c.reshape(-1)
print(d,h)                # [ 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  5  6  7  8  9 10 11 12] [ 1  2  3  4  5  6  7  8  9 10 11 12]
```

NumPy Array Indexing

Access 1D,2D,3D Elements

[59]:

```
barr = np.array([[1, 2, 3], [4, 5, 6]])
print(barr[0])
print(barr[0, 1])
print(barr[1, 2])
c=np.array([[[1, 2, 3], [4, 5, 6]]])
print(c[0, 1, 2])
```

```
[1 2 3]

2

6

6
```

Negative Indexing

[60]:

```
barr = np.array([[1, 2, 3], [4, 5, 6]])
print('Last element from 2nd dim: ', barr[1, -1])
```

```
Last element from 2nd dim:  6
```

Slicing arrays

[62]:

```
barr = np.array([[1, 2, 3], [4, 5, 6]])
print(barr)
print(barr[0, 1:3])
```

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

[2 3]
```

[64]:

```
carr1 = np.array([
 [ [1,2,3],[4,5,6],[7,8,9] ],
 [ [1,2,3],[4,5,6],[7,8,9] ],
 [ [1,2,3],[4,5,6],[7,8,9] ],
 ])
print(carr1[0][1][2]) #1st 2-D, 2nd row, 3rd element
print(carr1[0,1,2])
carr1[0, 0, 0] = 100
carr1[1, 0:2, 0:2] = 200
carr1[2, 0, 0] = 300
print(carr1)
print("Dimension of carr1 = ",carr1.ndim)
print(carr1.shape)
```

```
6
6
[[[100   2   3]
  [  4   5   6]
  [  7   8   9]]

 [[200 200   3]
  [200 200   6]
  [  7   8   9]]

 [[300   2   3]
  [  4   5   6]
  [  7   8   9]]]
Dimension of carr1 =  3
(3, 3, 3)
```

Broadcasting

[61]:

```
a=np.array([1,2,3])
a=np.array([1,2])
print(a+b)    # broadcasting error
```

```
---------------------------------------------------------------------------
```

```
ValueError                                Traceback (most recent call last)
Cell In[61], line 3
      1 a=np.array([1,2,3])
      2 a=np.array([1,2])
----> 3 print(a+b)     # broadcasting error

ValueError: operands could not be broadcast together with shapes (2,) (2,3)
```

[ ]:

```
a=np.array([1,2,3])
b=np.array([1,2,3])
print(a+b)     # [2 4 6]
```

```
[2 4 6]
```

Rules to remember while applying operation :

```
1. Same Dimension : compare dimension from right side
2. If we have two dimensions (1x3) and (3x1) so both should have "1" atleast in the end or both
```

Note : It will give output as (3x3) which is the maximum dimension from both the dimensions.

[ ]:

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

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

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

```
    Note : a has (1x3) and b has (3x1) dimension having 1 on any of two right hand side ,and re

    If a has (1x3) and b has (1x4) then it will through broadcasting error.
    If a has (1x3) and b has (3x4) then it will also through broadcasting error.
```

[ ]:

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

```
[[1]
 [2]]
```

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

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

Note : a has (2x1) and b has (2x3) dimension having 1 on any of two right hand side , and resul will be maximum dimension from both the dimensions i.e (2x3)

nditer() - use for fetching all element without using nested for loop

```
[ ]: a=np.array([[[1,2],[5,6]]])
     s=np.nditer(a)
     print(s)
     for i in s:
         print(i)
```

```
<numpy.nditer object at 0x000001ACFF4F6790>
1
2
5
6
```

```
[ ]: a=np.array([[[1,2],[5,6]]],dtype="S")   # here s stands for string
     for i in np.nditer(a):
         print(i)
```

```
b'1'
b'2'
b'5'
b'6'
```

ndenumerate() :- use to print index along with iteration

```
[ ]: a=np.array([[[1,2],[5,6]]])
     for i,d in np.ndenumerate(a):   # Gives index and value
         print(i,d)
```

```
(0, 0, 0) 1
(0, 0, 1) 2
(0, 1, 0) 5
(0, 1, 1) 6
```

copy() & view()

copy	view
The copy own the data	The view does not own the da
The copy of array i a new array	A view of original array
changes made in a copied array doesnot affect original one	Any changes in viewed or orig

[65]:

```
var=np.array([1,2,3])
c=var.copy()
print("var before :",var)
print("c before :",c)
c[0]=5
print("var after :",var)
print("c after :",c)
print(c.base)
```

```
var before : [1 2 3]
c before : [1 2 3]
var after : [1 2 3]
c after : [5 2 3]
None
```

[66]:

```
var=np.array([1,2,3])
c=var.view()
print("var before :",var)
print("c before :",c)
c[0]=5
print("var after :",var)
print("c after :",c)
print(c.base)
```

```
var before : [1 2 3]
c before : [1 2 3]
var after : [5 2 3]
c after : [5 2 3]
[5 2 3]
```

JOIN ARRAY

    concatenate()

[68]:

```
v1=np.array([1,2,3])
v2=np.array([3,4,5])
new=np.concatenate((v1,v2)) # Must be in tuple
print(new)
```

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

[72]:

```
v1=np.array([[1,2],[4,5]])
v2=np.array([[7,6],[9,15]])
n1=np.concatenate((v1,v2),axis=0)
n2=np.concatenate((v1,v2),axis=1)
print(n1)
print(n2)
```

```
[[ 1  2]
```

```
 [ 4  5]
 [ 7  6]
 [ 9 15]]

[[ 1  2  7  6]
 [ 4  5  9 15]]
```

```
stack()
```

[74]:
```
v1=np.array([1,2,3])
v2=np.array([3,4,5])
n1=np.stack((v1,v2),axis=0)
n2=np.stack((v1,v2),axis=1)
print(n1)
print(n2)
```

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

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

```
hstack() : along row
vstack() : along column
dtack() : along height
```

[80]:
```
v1=np.array([[1,2],[4,5]])
v2=np.array([[7,6],[9,15]])
n1=np.hstack((v1,v2))
n2=np.vstack((v1,v2))
n3=np.dstack((v1,v2))
print(n1)
print(n2)
print(n3)
```

```
[[ 1  2  7  6]
 [ 4  5  9 15]]

[[ 1  2]
 [ 4  5]
 [ 7  6]
 [ 9 15]]

[[[ 1  7]
  [ 2  6]]

 [[ 4  9]
  [ 5 15]]]
```

SPLIT ARRAY

[89]:
```
v1=np.array([1,2,4,5])
n1=np.array_split(v1,1)
n2=np.array_split(v1,2)
n3=np.array_split(v1,3)
n4=np.array_split(v1,4)
print(n1)
print(n2)
print(n3)
print(n4)
# Here it will give data as list
```

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

[96]:
```
v2=np.array([[1,2,3,4],[5,6,7,8]])
n5=np.array_split(v2,4,axis=0)
n6=np.array_split(v2,4,axis=1)
print(n5)
print(n6)
```

```
[array([[1, 2, 3, 4]]), array([[5, 6, 7, 8]]), array([], shape=(0, 4),
dtype=int32), array([], shape=(0, 4), dtype=int32)]
[array([[1],
       [5]]), array([[2],
       [6]]), array([[3],
       [7]]), array([[4],
       [8]])]
```

SEARCH ARRAY

[100]:
```
v1=np.array([1,2,4,5])
x=np.where(v1%2==0)
y=np.where(v1==4)
print(x)
print(y)
```

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

SEARCH SORTED ARRAY :
    Performs a binary search in the array and return indexes where the specified value would b

[106]:
```
var=np.array([1,2,4,5,8,9])
x=np.searchsorted(var,3)
print(x)
y=np.searchsorted(var,9)
```

```
print(y)
```

```
2
5
```

[108]:
```
var=np.array([1,2,4,5,8,9])
x=np.searchsorted(var,[3,5,2])
print(x)
```

```
[2 3 1]
```

side (parameter)

[107]:
```
var=np.array([1,2,4,5,8,9])
x=np.searchsorted(var,3,side="right")
print(x)
y=np.searchsorted(var,9,side="right")
print(y)
```

```
2
6
```

SORT ARRAY

[111]:
```
var=np.array([[32,4,56],[3,5,1]])
print(np.sort(var))
```

```
[[ 4 32 56]
 [ 1  3  5]]
```

FILTER ARRAY

[114]:
```
var=np.array([1,2,3])
f=[True,False,True]
p=[1,0,1]
n1=var[f]
n2=var[p]
print(n1,n2)
```

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

[115]:
```
var=np.array([1,2,3,4,5,6,7,9])
print(var[var%2==0])
```

```
[2 4 6]
```

INSERT

[133]:
```
var=np.array([1,2,3,4])
n1=np.insert(var,1,24)
print(n1)
```

```
n2=np.insert(var,[3,4],24.5)
print(n2) # it will not accept floating value so it convert it into integer
```

```
[ 1 24  2  3  4]
[ 1  2  3 24  4 24]
```

[132]:

```
var=np.array([[1,2],[3,4]])
n1=np.insert(var,2,24,axis=0)
n2=np.insert(var,2,24,axis=1)
n3=np.insert(var,2,[24,22],axis=1)
n4=np.insert(var,2,[24,22],axis=1)
print(n1)
print(n2)
print(n3)
print(n4)
```

```
[[ 1  2]
 [ 3  4]
 [24 24]]

[[ 1  2 24]
 [ 3  4 24]]

[[ 1  2 24]
 [ 3  4 22]]

[[ 1  2 24]
 [ 3  4 22]]
```

APPEND

[138]:

```
var=np.array([1,2,3,4])
n1=np.append(var,24)
print(n1)
n2=np.append(var,24.5)
print(n2)
```

```
[ 1  2  3  4 24]
[ 1.   2.   3.   4.  24.5]
```

DELETE

[141]:

```
var=np.array([1,2,3,4])
d1=np.delete(var,2)
print(d1)
d2=np.delete(var,[2,0])
print(d2)
```

```
[1 2 4]
```

```
[2 4]
```

MATRIX

[142]:

```
var1=np.matrix([[1,2,3],[3,4,5]])
print(var1)
print(var1+var1)
```

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

[[ 2  4  6]
 [ 6  8 10]]
```

[165]:

```
var1=np.matrix([[1,2],[3,4]])
var2=np.matrix([[1,2],[3,4]])
print(var1.dot(var2)) # we can use " * " instead of dot
# Must follow broadcast rule
```

```
[[ 7 10]
 [15 22]]
```

transpose()

[164]:

```
var=np.matrix([[1,2],[3,4]])
print(np.transpose(var))
```

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

Another way of finding transpose

[166]:

```
var=np.matrix([[1,2],[3,4]])
print(var.T)
```

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

swapaxes()

[167]:

```
var=np.matrix([[1,2],[3,4]])
print(np.swapaxes(var,1,0))
```

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

Inverse of Matrix

[161]:

```
var=np.matrix([[1,2],[3,4]])
print(np.linalg.inv(var))
```

```
[[-2.   1. ]
 [ 1.5 -0.5]]
```

power

```
[173]: var=np.matrix([[1,2],[3,4]])
       print(np.linalg.matrix_power(var,2)) # if we put 0 then we get identity matrix
```

```
[[ 7 10]
 [15 22]]
```

Determinant

```
[179]: var=np.matrix([[1,2],[3,8]])
       print(np.linalg.det(var))
```

```
1.9999999999999998
```

```
[ ]:
```

```
[ ]:
```