Numpy

SURAJ SUTAR

```
Numpy:-(Numeric python),it is an library in python which is deals with arrays
```

In [ ]:
```
1] be an data scientiest it is important library.
2] we have to work with different different array 1d,2d,3d...nd it helps for that
```

In [ ]:
```
Application:-
    1]linear algebra
    2]random number generation
    3]matrix calculation
    4]fourier transfrom
    5]statistical problem
    6]lagrithimic
    7]trignometry
    8]image processing
```

In [ ]:
```
Properties of Numpy:-
    1] It consumes less memory than python sequence datatype
    2] It is faster than python sequence datatype
    3] By using its vast application we can solve the problems easily
    4] It is homogeneous datatype
    5] It provides efficient storage
    6] It also provide better way to handling data
```

In [ ]:
```
Array creation in numpy:-
    1] By converting python sequence datatype into array (eg.list,tuple,set)
    2] By using existing numpy functions(arange,reshape,full,zeros,ones,linspace,empty)
    3] Replacing,joining existing array
    4] Reading array from disk either from standard or custom fromat
    5] Use special library function
```

In [ ]:
```
for using numpy library you have to install numpy library you can install it by:-
    pip install numpy
    (pip :- package installer python)

we can get numpy by importing using :-
    import numpy as np
    >>>> we are creating alias name of numpy by using as and we put the alias name as the np.
    >>>> because np sounds like a np.

```

In [3]:
```
import numpy as np
```

In [ ]:
```
datatype:-datatype of numpy is >>>> numpy.ndarray <<<<
```

In [ ]:
```
nd array:-
    1] we can create n-dimensional array in python
    2] but as a data scientiest we required upto only 3d array
```

In [ ]:

```
Numpy Functions:-
    1]array.ndim
    2]array.ndmin
    3]array.shape
    4]array.reshape
    5]np.nditer
    6]np.ndenumerate
    7]np.copy
    8]np.append
    9]np.concatenate
    10]np.zeros
    11]np.delete
    12]np.ones
    13]np.full
    14]np.linspace
    15]np.where
    16]np.intersect1d
    17]array.argmax
    18]array.argmin
    19]np.argsort
    20]np.sort
    21]np.flip
    22]axis=0,1(1=for columns,0=for rows)
    23]np.eye
    24]np.identity
    25]np.filldiagonal
    26]arr.nbytes
    27]arr.itemsize

mathmatical function:-
    1]sum
    2]multiply
    3]add
    4]dot

random function:
    1] random.rand
    2] random.randint
    3] random.randn
    4] random.choice

trignometry:
    1]np.sin
    2]np.cos
    3]np.tan
    4]np.deg2rad
    5]np.rad2deg

linear algebra:-
    1]np.linalg.solve
    2]np.linslg.inv

statistical calculation:-
    1]np.mean
    2]np.median
    3]np.var
    4]nd.std
    5]st.mode
logrithimic:-
    1]np.log
    2]np.log2
    3]np.log10
```

1D Array:-

In [3]:

```
arr=np.array([1,2,3,4,5,6,7])
print("Original array:-\n",arr)
print("Dimensions:-",arr.ndim)
print("DataType of array:-",type(arr))
print("Datatype of elements into it:-",arr.dtype)
```

```
Original array:-
 [1 2 3 4 5 6 7]
Dimensions:- 1
DataType of array:- <class 'numpy.ndarray'>
Datatype of elements into it:- int32
```

2D Array:-

In [4]:
```
arr=np.array([[1,2,3,4,5],[1,2,3,4,5]])
print("Original array:-\n",arr)
print("Dimensions:-",arr.ndim)
print("DataType of array:-",type(arr))
print("Datatype of elements into it:-",arr.dtype)
```

```
Original array:-
 [[1 2 3 4 5]
 [1 2 3 4 5]]
Dimensions:- 2
DataType of array:- <class 'numpy.ndarray'>
Datatype of elements into it:- int32
```

3d array:-

In [5]:
```
arr=np.array([[[11,22,33],[44,55,66]],[[1,2,3],[4,5,6]]])
print("Original array:-\n",arr)
print("Dimensions:-",arr.ndim)
print("DataType of array:-",type(arr))
print("Datatype of elements into it:-",arr.dtype)
```

```
Original array:-
 [[[11 22 33]
  [44 55 66]]
 [[ 1  2  3]
  [ 4  5  6]]]
Dimensions:- 3
DataType of array:- <class 'numpy.ndarray'>
Datatype of elements into it:- int32
```

In [ ]:
```
Note:- keep the array size with same lengths otherwise it gives VisibleDeprecationWarning
```

In [7]:
```
arr=np.array([[[11,22,33],[44,55,66]],[[2,3],[4,5,6]]])
print(arr)
```

```
[[list([11, 22, 33]) list([44, 55, 66])]
 [list([2, 3]) list([4, 5, 6])]]

C:\Users\hp\AppData\Local\Temp\ipykernel_16616\2711129163.py:1: VisibleDeprecationWarning: Creating an ndarray from rag
ged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is depr
ecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray.
  arr=np.array([[[11,22,33],[44,55,66]],[[2,3],[4,5,6]]])
```

Array slicing:-

In [8]:
```
a=np.array([[11,22,33],[44,55,66],[77,88,99]])
print(a)
```

```
[[11 22 33]
 [44 55 66]
 [77 88 99]]
```

In [9]:
```
a[0]
```

Out[9]:
```
array([11, 22, 33])
```

In [10]:
```
a[0:2]
```

Out[10]:
```
array([[11, 22, 33],
       [44, 55, 66]])
```

In [11]:
```
a[0:,0] #it will gives first column
```

Out[11]:
```
array([11, 44, 77])
```

In [13]:
```
a[0:,0:2] #it will gives starting two columns
```

Out[13]:
```
array([[11, 22],
       [44, 55],
       [77, 88]])
```

In [14]:
```
a[2,1:]
```

Out[14]:
```
array([88, 99])
```

ndmin()

In [ ]:
```
>> We can change the dimension of an array by using the ndmin function
>> We can only add the dimension we cant get it lower
```

In [9]:
```
arr=np.array([1,2,3,4])
arr.ndim
```

Out[9]: 1

In [10]:
```
arr=np.array([1,2,3,4],ndmin=8)#here we are changing the dimensions of it
arr.ndim
```

Out[10]: 8

Numpy Functions

arr.itemsize

In [190]:
```
#It will give the size of single element in the array in bytes
arr=np.array([[1,2,3],[4,5,6]])
arr.itemsize #Every datatype size will be different
```

Out[190]: 4

arr.nbytes

In [196]:
```
#It will give the size of whole array in bytes
arr=np.array([[1,2,3],[4,5,6]])
print("Size of whole array:-",arr.nbytes)
print("Size of sigle element:-",arr.itemsize)
print("Data Type of an elements:-",arr.dtype)
```

```
Size of whole array:- 24
Size of sigle element:- 4
Data Type of an elements:- int32
```

In [195]:
```
arr=np.array([[1.3,2,3],[4,5,6]])
print("Size of whole array:-",arr.nbytes)
print("Size of sigle element:-",arr.itemsize)
print("Data Type of an elements:-",arr.dtype) #all values gets float here apply typecasting rule
```

```
Size of whole array:- 48
Size of sigle element:- 8
Data Type of an elements:- float64
```

np.delete()

In [ ]:
```
>> It is use for deleting specific row and column
```

In [198]:
```
a=np.array([[1,2,3,4],[5,6,7,8]])
np.delete(a,0,axis=1) #It gets deleted first column
```

Out[198]:
```
array([[2, 3, 4],
       [6, 7, 8]])
```

In [199]:
```
a=np.array([[1,2,3,4],[5,6,7,8]])
np.delete(a,0,axis=0) #It gets deleted first Row
```

Out[199]:
```
array([[5, 6, 7, 8]])
```

shape()

In [ ]:
```
>> If we want to check shape of an array then we can use this fucntion
>> It will display the How many rows and columns are present in the array
```

In [13]:
```
a=np.array([[11,22,33],[44,55,66],[77,88,99]])
print(a)
a.shape #it contains three rows and three columns
```

```
[[11 22 33]
 [44 55 66]
 [77 88 99]]
```

Out[13]: (3, 3)

In [14]:
```
#checking it by changing the dimension of it
a=np.array([[11,22,33],[44,55,66],[77,88,99]],ndmin=5)
print(a)
a.shape
```

```
[[[[[11 22 33]
    [44 55 66]
    [77 88 99]]]]]
```

Out[14]: (1, 1, 1, 3, 3)

reshape()

In [ ]:
```
>> we can change the shape or dimension of an array by this function
>> NOTE:- size will be fit in that pattern
```

In [23]:
```
a=np.array([[11,22,33],[44,55,66],[77,88,99],[11,22,33]])
print("Original array:-\n",a)
print("Size of an array:-",a.size)
print("Dimension of an array:-",a.ndim)
print("Shape of an array:-",a.shape)
print("*"*80)
print("We are cahanging shape of an array:-\n",a.reshape(2,2,3))
print('Array shape after reshape:-\n',a.shape)
print("Dimension of an array:-",a.ndim)
```

```
Original array:-
 [[11 22 33]
 [44 55 66]
 [77 88 99]
 [11 22 33]]
Size of an array:- 12
Dimension of an array:- 2
Shape of an array:- (4, 3)
********************************************************************************
We are cahanging shape of an array:-
 [[[11 22 33]
  [44 55 66]]

 [[77 88 99]
  [11 22 33]]]
Array shape after reshape:-
 (4, 3)
Dimension of an array:- 2
```

size()

In [24]:
```
a=np.array([[11,22,33],[44,55,66],[77,88,99],[11,22,33]])
print("Size of an array:-",a.size)
```

```
Size of an array:- 12
```

np.nditer()

In [ ]:
```
>> it is used to iterate through the array
```

In [27]:
```
a=np.nditer([[11,22,33],[44,55,66],[77,88,99],[11,22,33]])
for i in a:
    print(i)
```

```
(array(11), array(44), array(77), array(11))
(array(22), array(55), array(88), array(22))
(array(33), array(66), array(99), array(33))
```

np.ndenumerate()

In [ ]:
```
>> it will display the value and the index number of that value
```

In [28]:
```
a=np.ndenumerate([[11,22,33],[44,55,66],[77,88,99],[11,22,33]])
for i,j in a:
    print(f"index is:-{i},value is:-{j}")
```

```
index is:-(0, 0),value is:-11
index is:-(0, 1),value is:-22
index is:-(0, 2),value is:-33
index is:-(1, 0),value is:-44
index is:-(1, 1),value is:-55
index is:-(1, 2),value is:-66
index is:-(2, 0),value is:-77
index is:-(2, 1),value is:-88
index is:-(2, 2),value is:-99
index is:-(3, 0),value is:-11
index is:-(3, 1),value is:-22
index is:-(3, 2),value is:-33
```

flatten()

In [ ]:
```
>> it gets flatten the array
```

In [32]:
```
a=np.array([[11,22,33],[44,55,66],[77,88,99],[11,22,33]])
print("Original array:-\n",a)
print("Array after flatten:-",a.flatten())
```

```
Original array:-
 [[11 22 33]
 [44 55 66]
 [77 88 99]
 [11 22 33]]
Array after flatten:- [11 22 33 44 55 66 77 88 99 11 22 33]
```

ravel()

In [ ]:
```
>> it also gets flatten the array
```

In [39]:
```
a=np.array([[11,22,33],[44,55,66],[77,88,99],[11,22,33]])
b=a
print(a)
b[0,0]=99
print("Flattening the array:-",b.ravel())
print(a)
#here we observe that if we make changes into the copied array then it also gets implemented into original
#For that we have to use the deepcopy()
```

```
[[11 22 33]
 [44 55 66]
 [77 88 99]
 [11 22 33]]
Flattening the array:- [99 22 33 44 55 66 77 88 99 11 22 33]
[[99 22 33]
 [44 55 66]
 [77 88 99]
 [11 22 33]]
```

copy()

In [44]:
```
a=np.array([[11,22,33],[44,55,66],[77,88,99],[11,22,33]])
b=a
print("Original array:-\n",a)
b[0,0]=99 #changing element
print("Copied array:-\n",b)
print("Original array:-\n",a)
```

```
Original array:-
 [[11 22 33]
 [44 55 66]
 [77 88 99]
 [11 22 33]]
Copied array:-
 [[99 22 33]
 [44 55 66]
 [77 88 99]
 [11 22 33]]
Original array:-
 [[99 22 33]
 [44 55 66]
 [77 88 99]
 [11 22 33]]
```

In [45]:
```
a=np.array([[11,22,33],[44,55,66],[77,88,99],[11,22,33]])
b=a.copy() #using copy function
print("Original array:-\n",a)
b[0,0]=99 #changing element
print("Copied array:-\n",b)
print("Original array:-\n",a)
```

```
Original array:-
 [[11 22 33]
 [44 55 66]
 [77 88 99]
 [11 22 33]]
Copied array:-
 [[99 22 33]
 [44 55 66]
 [77 88 99]
 [11 22 33]]
Original array:-
 [[11 22 33]
 [44 55 66]
 [77 88 99]
 [11 22 33]]
```

np.linspace()

In [ ]:
```
>> It gives linearly spaced elements into that range
>> If we cant give the value then by default it will give 50 values
>> Syntax:-
    np.linspace(start_index,end_index,values,retstep=False)
```

In [48]:
```
arr=np.linspace(10,20,retstep=True)
print(arr) #by default it is given the 50 values
```

```
(array([10.        , 10.20408163, 10.40816327, 10.6122449 , 10.81632653,
       11.02040816, 11.2244898 , 11.42857143, 11.63265306, 11.83673469,
       12.04081633, 12.24489796, 12.44897959, 12.65306122, 12.85714286,
       13.06122449, 13.26530612, 13.46938776, 13.67346939, 13.87755102,
       14.08163265, 14.28571429, 14.48979592, 14.69387755, 14.89795918,
       15.10204082, 15.30612245, 15.51020408, 15.71428571, 15.91836735,
       16.12244898, 16.32653061, 16.53061224, 16.73469388, 16.93877551,
       17.14285714, 17.34693878, 17.55102041, 17.75510204, 17.95918367,
       18.16326531, 18.36734694, 18.57142857, 18.7755102 , 18.97959184,
       19.18367347, 19.3877551 , 19.59183673, 19.79591837, 20.        ]), 0.20408163265306123)
```

In [49]:
```
arr=np.linspace(10,20,15,retstep=True)
print(arr)
```

```
(array([10.        , 10.71428571, 11.42857143, 12.14285714, 12.85714286,
       13.57142857, 14.28571429, 15.        , 15.71428571, 16.42857143,
       17.14285714, 17.85714286, 18.57142857, 19.28571429, 20.        ]), 0.7142857142857143)
```

np.arange()

In [ ]:
```
>> if will give the range between that number
>> Syntax:-
    np.arange(start.index,end.index)
```

In [50]:
```
arr=np.arange(10,20)
print(arr)
```

```
[10 11 12 13 14 15 16 17 18 19]
```

In [ ]:
```
we can also change shape of it
```

In [51]:
```
arr=np.arange(10,22).reshape(4,3)
print(arr)
```

```
[[10 11 12]
 [13 14 15]
 [16 17 18]
 [19 20 21]]
```

np.eye()

In [ ]:
```
>> It will give identity matrix but we can change the shape of it
>> we can also shift the diagonal elements of the array
```

In [52]:
```
arr=np.eye(4,4)
print(arr)
```

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

In [53]:
```
arr=np.eye(4,4,k=-1) #changing row elements downside
print(arr)
```

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

np.identity()

In [ ]:
```
>> it gives the identity matrix
```

In [55]:
```
arr=np.identity(4)
print(arr)
```

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

np.full()

In [ ]:
```
>> It gets full the array with number what number we want to fill
>> Syntax:-
    np.full((ndarray),number)

```

In [57]:
```
arr=np.full((5,4),8)
print(arr)
```

```
[[8 8 8 8]
 [8 8 8 8]
 [8 8 8 8]
 [8 8 8 8]
 [8 8 8 8]]
```

np.zeros()

In [ ]:
```
>>> It returns the array with zeros
```

In [59]:
```
arr=np.zeros((4,4))
print(arr)
```

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

np.ones()

In [ ]:
```
>>> It returns the array with one
```

In [60]:
```
arr=np.ones((4,4))
print(arr)
```

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

np.fill_diagonal()

In [66]:
```
arr=np.ones((4,4))
print(arr)
print("*"*80)
np.fill_diagonal(arr,9)
print(arr)
```

```
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
********************************************************************************
[[9. 1. 1. 1.]
 [1. 9. 1. 1.]
 [1. 1. 9. 1.]
 [1. 1. 1. 9.]]
```

np.append()

In [ ]:
```
>> it is used to join the two array
>> Axis:-
    axis=0 for row
    axis=1 for columns
```

In [67]:
```
arr=np.ones((4,4))
arr1=np.full((4,4),5)
print(np.append(arr,arr1,axis=0))
```

```
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [5. 5. 5. 5.]
 [5. 5. 5. 5.]
 [5. 5. 5. 5.]
 [5. 5. 5. 5.]]
```

In [68]:
```
arr=np.ones((4,4))
arr1=np.full((4,4),5)
print(np.append(arr,arr1,axis=1))
```

```
[[1. 1. 1. 1. 5. 5. 5. 5.]
 [1. 1. 1. 1. 5. 5. 5. 5.]
 [1. 1. 1. 1. 5. 5. 5. 5.]
 [1. 1. 1. 1. 5. 5. 5. 5.]]
```

np.concatenate()

In [ ]:
```
>> It is used to join the n no of arrays
```

In [70]:
```
arr=np.ones((4,4))
arr1=np.full((4,4),5)
arr2=np.full((4,4),8)
print(np.concatenate((arr,arr1,arr2),axis=1))
```

```
[[1. 1. 1. 1. 5. 5. 5. 5. 8. 8. 8. 8.]
 [1. 1. 1. 1. 5. 5. 5. 5. 8. 8. 8. 8.]
 [1. 1. 1. 1. 5. 5. 5. 5. 8. 8. 8. 8.]
 [1. 1. 1. 1. 5. 5. 5. 5. 8. 8. 8. 8.]]
```

arr.tolist()

In [ ]:
```
>> It is used to convert array into list
```

In [180]:
```
arr1=np.full((4,4),5)
print(arr1)
print(type(arr1))
```

```
[[5 5 5 5]
 [5 5 5 5]
 [5 5 5 5]
 [5 5 5 5]]
<class 'numpy.ndarray'>
```

In [183]:
```
arr2=arr1.tolist()
print(arr2)
print(type(arr2))
```

```
[[5, 5, 5, 5], [5, 5, 5, 5], [5, 5, 5, 5], [5, 5, 5, 5]]
<class 'list'>
```

np.argmax()

In [ ]:
```
>> It will give index number of maximum number
>> It will faltten the array and give the index number
```

In [186]:
```
a=np.random.randint(10,80,(4,4))
print(a)
print("Index number of maximun argument:-",np.argmax(a))
```

```
[[37 27 54 14]
 [31 39 41 65]
 [22 31 64 43]
 [44 39 36 39]]
Index number of maximun argument:- 7
```

np.argmin()

In [ ]:
```
>> It will give index number of minimum number
>> It will faltten the array and give the index number
```

In [188]:
```
a=np.random.randint(10,80,(4,4))
print(a)
print("Index number of minimum argument:-",np.argmin(a))
```

```
[[20 27 47 44]
 [47 60 28 79]
 [29 53 21 76]
 [53 54 12 15]]
Index number of minimum argument:- 14
```

np.argsort()

In [ ]:
```
>> It will says how to sort the array by index
```

In [189]:
```
a=np.random.randint(10,80,(4,4))
print(a)
print(np.argsort(a))
```

```
[[12 68 77 77]
 [39 51 69 74]
 [58 24 26 62]
 [15 69 15 58]]
[[0 1 2 3]
 [0 1 2 3]
 [1 2 0 3]
 [0 2 3 1]]
```

np.sort()

In [ ]:
```
>> It is used to sort an array
```

In [71]:
```
a=np.random.randint(10,60,(5,4))
print(a)
```

```
[[15 39 10 18]
 [17 24 11 49]
 [22 59 55 18]
 [28 40 46 55]
 [54 40 23 58]]
```

In [72]:
```
print(np.sort(a))#it is sorting by row wise
```

```
[[10 15 18 39]
 [11 17 24 49]
 [18 22 55 59]
 [28 40 46 55]
 [23 40 54 58]]
```

In [75]:
```
print(np.sort(a,axis=0)) #it gets sorted column wise
```

```
[[15 24 10 18]
 [17 39 11 18]
 [22 40 23 49]
 [28 40 46 55]
 [54 59 55 58]]
```

np.flip()

In [76]:
```
a=np.random.randint(10,60,(5,4))
print("Original Array:-\n",a)
```

```
Original Array:-
 [[59 56 50 22]
 [54 11 56 43]
 [37 57 53 42]
 [59 39 51 17]
 [55 52 32 51]]
```

In [77]:
```
#Fliping the array
print(np.flip(a))
```

```
[[51 32 52 55]
 [17 51 39 59]
 [42 53 57 37]
 [43 56 11 54]
 [22 50 56 59]]
```

np.intersect1d()

In [78]:
```
a=np.random.randint(10,60,(5,4))
b=np.random.randint(10,60,(5,4))
```

In [79]:
```
print(np.intersect1d(a,b)) #it gives common elements between both the array
```

```
[22 31 34 42 49]
```

np.size()

In [163]:
```
#It gives how many elements present in the array size of an array
a=np.random.randint(10,60,(5,4))
print(a.size)
```

```
20
```

np.where()

In [ ]:
```
>>> it is used to find the conditions
```

In [80]:
```
a=np.random.randint(10,60,(5,4))
print(a)
```

```
[[25 12 15 38]
 [52 13 14 39]
 [58 45 55 19]
 [31 42 15 46]
 [11 49 24 57]]
```

In [81]:
```
print(np.where(a==55))
```

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

np.empty()

In [ ]:
```
>> it will fill the elements with the random elements
```

In [86]:
```
arr=np.empty((5,5),dtype=int)
print(arr)
```

```
[[ 975636904        720         64          0          0]
 [         0          0          0          0    6881382]
 [1630627430 1630561330  895706214  859202869  845493859]
 [ 825768290  946092338 1714512439  845231668 1681285426]
 [ 812005431 1630954545 1664640613  959854129  875771447]]
```

Random number generation:------

In [ ]:
```
we are use following functions for random number generation:-
    1] np.random.rand()
    2] np.random.randint()
    3] np.random.randn()
    4] np.random.choice()
    5] np.random.ranf()
```

np.random.rand()

In [ ]:
```
>> It gives the random numbers between 0 to 1
```

In [87]:
```
arr=np.random.rand(4,4)
print(arr)
```

```
[[0.79957277 0.31597598 0.82660681 0.71905827]
 [0.55541956 0.8824127  0.00377776 0.61257476]
 [0.51624343 0.97808491 0.47922011 0.39264564]
 [0.23026611 0.97381071 0.00212077 0.37642903]]
```

In [88]:
```
# we can also round the values upto what digit we have
arr=np.random.rand(4,4).round(2)
print(arr)
```

```
[[0.8  0.54 0.97 0.43]
 [0.26 0.77 0.54 0.32]
 [0.4  0.05 0.65 0.74]
 [0.85 0.04 0.14 0.76]]
```

np.random.randint()

In [ ]:
```
>> It is used to create the random numbers between that ranges
>> we can also mention the shape it what shape we want the array
>> Syntax:-
    np.random.randint(start_index,end_index,(shape))
```

In [89]:
```
arr=np.random.randint(10,20,(5,4))
print(arr)
```

```
[[15 19 18 10]
 [12 11 12 13]
 [19 12 19 17]
 [11 18 14 10]
 [16 19 18 19]]
```

np.random.randn()

In [ ]:
```
>> It gives the array with the random numbers
```

In [94]:
```
arr=np.random.randn(4,4).round(2)
print(arr)
```

```
[[-0.45 -2.09  1.86 -0.8 ]
 [-0.89 -0.56  0.44  1.83]
 [ 1.61 -1.03 -1.34 -0.17]
 [-1.35  0.94 -0.49  0.82]]
```

np.random.choice()

In [ ]:
```
>> We have to enter the choice for filling the array full of it
>> Syntax:-
    np.random.choice((choices),(shape))
```

In [102]:
```
arr=np.random.choice((1,2,3),(4,4))
print(arr)
```

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

np.random.ranf()

In [ ]:
```
>> It is same as like np.random.rand()
```

Statistical:-

In [ ]:
```
1] np.mean():-It the avg value of the all the value
2] np.median() :- It is the centre value of the sorted array
3] np.std() :- It is the squareroot of variance
4] np.var() :- It is the summation of (val+mean)/n
for mode we have to from scipy.stats import mode
5]stats.mode():- most occured value
```

np.mean()

In [112]:
```
a=np.random.randint(10,20,(5,4))
print(a)
print("Mean is:-",np.mean(a)) #it gets mean of whole array
```

```
[[17 17 11 17]
 [19 18 18 16]
 [14 12 10 19]
 [15 19 10 14]
 [11 18 18 13]]
Mean is:- 15.3
```

In [113]:
```
#axis wise mean
print("axis 0 Mean is:-",np.mean(a,axis=0))
print("axis 1 Mean is:-",np.mean(a,axis=1))
```

```
axis 0 Mean is:- [15.2 16.8 13.4 15.8]
axis 1 Mean is:- [15.5  17.75 13.75 14.5  15.  ]
```

np.median()

In [115]:
```
a=np.random.randint(10,20,(5,4))
print(a)
print(np.median(a,axis=1))
```

```
[[19 10 17 18]
 [17 11 14 19]
 [15 13 10 17]
 [15 14 13 17]
 [14 16 17 12]]
[17.5 15.5 14.  14.5 15. ]
```

In [116]:
```
print(np.median(a))
```

```
15.0
```

In [117]:
```
print(np.median(a,axis=0))
```

```
[15. 13. 14. 17.]
```

In [118]:
```
print(np.median(a,axis=1))
```

```
[17.5 15.5 14.  14.5 15. ]
```

np.std()

In [119]:
```
a=np.random.randint(10,20,(5,4))
print(a)
print(np.std(a,axis=1))
```

```
[[19 12 14 15]
 [19 17 17 13]
 [12 17 12 11]
 [16 16 14 11]
 [16 12 18 10]]
[2.54950976 2.17944947 2.34520788 2.04633819 3.16227766]
```

In [120]:
```
print(np.std(a,axis=0))
```

```
[2.57681975 2.31516738 2.19089023 1.78885438]
```

In [121]:
```
print(np.std(a))
```

```
2.747271373563231
```

np.var()

In [122]:
```
a=np.random.randint(10,20,(5,4))
print(a)
print(np.var(a,axis=1))
```

```
[[18 19 11 11]
 [12 12 12 19]
 [14 15 15 18]
 [14 11 14 14]
 [12 19 12 14]]
[14.1875  9.1875  2.25    1.6875  8.1875]
```

In [123]:
```
a=np.random.randint(10,20,(5,4))
print(a)
print(np.var(a,axis=0))
```

```
[[10 19 18 16]
 [17 14 12 17]
 [16 19 19 17]
 [17 18 17 13]
 [10 14 14 16]]
[10.8   5.36  6.8   2.16]
```

In [124]:
```
a=np.random.randint(10,20,(5,4))
print(a)
print(np.var(a))
```

```
[[15 18 13 19]
 [10 11 18 19]
 [16 18 16 12]
 [17 10 10 12]
 [15 11 19 12]]
10.747499999999999
```

stats.mode()

In [125]:
```
from scipy import stats
```

In [128]:
```
a=np.random.randint(10,20,(5,4))
print(stats.mode(a,axis=0))
```

```
ModeResult(mode=array([[19, 10, 18, 12]]), count=array([[2, 2, 2, 1]]))

C:\Users\hp\AppData\Local\Temp\ipykernel_13380\3342840602.py:2: FutureWarning: Unlike other reduction functions (e.g. `
skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, this be
havior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken wil
l be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  print(stats.mode(a,axis=0))
```

In [129]:
```
a=np.random.randint(10,20,(5,4))
print(stats.mode(a,axis=1))
```

```
ModeResult(mode=array([[10],
       [16],
       [15],
       [10],
       [11]]), count=array([[2],
       [2],
       [1],
       [1],
       [1]]))

C:\Users\hp\AppData\Local\Temp\ipykernel_13380\2580753765.py:2: FutureWarning: Unlike other reduction functions (e.g. `
skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, this be
havior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken wil
l be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  print(stats.mode(a,axis=1))
```

Trignometry

In [ ]:
```
We can use the Trignometric functions in the numpy:-
    1]np.sin()
    2]np.cos()
    3]np.tan()
    4]np.deg2rad()
    5]np.rad2deg()
```

In [133]:
```
a=np.random.randint(10,50,(4,4))
print("Original Array:-\n",a)
print(np.sin(a).round(2))
```

```
Original Array:-
 [[38 44 45 10]
 [20 16 46 30]
 [28 39 19 35]
 [11 49 35 24]]
[[ 0.3   0.02  0.85 -0.54]
 [ 0.91 -0.29  0.9  -0.99]
 [ 0.27  0.96  0.15 -0.43]
 [-1.   -0.95 -0.43 -0.91]]
```

In [134]:
```
a=np.random.randint(10,50,(4,4))
print("Original Array:-\n",a)
print(np.cos(a).round(2))
```

```
Original Array:-
 [[23 38 25 42]
 [48 39 21 17]
 [35 13 30 21]
 [24 12 28 20]]
[[-0.53  0.96  0.99 -0.4 ]
 [-0.64  0.27 -0.55 -0.28]
 [-0.9   0.91  0.15 -0.55]
 [ 0.42  0.84 -0.96  0.41]]
```

In [135]:
```
a=np.random.randint(10,50,(4,4))
print("Original Array:-\n",a)
print(np.tan(a).round(2))
```

```
Original Array:-
 [[48 23 39 25]
 [23 40 20 19]
 [23 16 14 45]
 [20 24 28 35]]
[[ 1.2   1.59  3.61 -0.13]
 [ 1.59 -1.12  2.24  0.15]
 [ 1.59  0.3   7.24  1.62]
 [ 2.24 -2.13 -0.28  0.47]]
```

In [136]:
```
a=np.random.randint(10,50,(4,4))
print("Original Array:-\n",a)
print(np.rad2deg(a).round(2))
```

```
Original Array:-
 [[17 33 38 32]
 [12 12 22 45]
 [15 42 44 32]
 [49 47 32 48]]
[[ 974.03 1890.76 2177.24 1833.46]
 [ 687.55  687.55 1260.51 2578.31]
 [ 859.44 2406.42 2521.01 1833.46]
 [2807.49 2692.9  1833.46 2750.2 ]]
```

In [137]:
```
a=np.random.randint(10,50,(4,4))
print("Original Array:-\n",a)
print(np.deg2rad(a).round(2))
```

```
Original Array:-
 [[37 44 38 23]
 [33 21 39 11]
 [48 36 26 28]
 [23 24 32 48]]
[[0.65 0.77 0.66 0.4 ]
 [0.58 0.37 0.68 0.19]
 [0.84 0.63 0.45 0.49]
 [0.4  0.42 0.56 0.84]]
```

Linear Algebra:-----

Quadratic equation

In [142]:
```
a=np.array([[1,2],[3,4]])
b=np.array([5,4])
print(np.linalg.solve(a,b))
```

```
[-6.   5.5]
```

Inverse of matrix

In [146]:
```
a=np.array([[1,2],[4,5]])
print(np.linalg.inv(a).round(2))
```

```
[[-1.67  0.67]
 [ 1.33 -0.33]]
```

Determinant of matrix

In [147]:
```
a=np.array([[1,2],[4,5]])
print(np.linalg.det(a).round(2))
```

```
-3.0
```

Logrithimic equations

In [ ]:
```
>> In lograthemic equation we can only use the log, log2, lg10
```

In [148]:
```
np.log(20)
```

Out[148]:
```
2.995732273553991
```

In [150]:
```
np.log2(16)
```

Out[150]:
```
4.0
```

In [151]:
```
np.log10(100)
```

Out[151]:
```
2.0
```

In [174]:
```
a=np.random.randint(10,20,(4,4))
print(a)
```

```
[[10 13 10 16]
 [14 13 14 17]
 [18 19 12 14]
 [11 11 16 10]]
```

In [176]:
```
print(np.log2(a).round(2))
```

```
[[3.32 3.7  3.32 4.  ]
 [3.81 3.7  3.81 4.09]
 [4.17 4.25 3.58 3.81]
 [3.46 3.46 4.   3.32]]
```

In [177]:
```
print(np.log10(a).round(2))
```

```
[[1.   1.11 1.   1.2 ]
 [1.15 1.11 1.15 1.23]
 [1.26 1.28 1.08 1.15]
 [1.04 1.04 1.2  1.  ]]
```

In [178]:
```
print(np.log(a).round(2))
```

```
[[2.3  2.56 2.3  2.77]
 [2.64 2.56 2.64 2.83]
 [2.89 2.94 2.48 2.64]
 [2.4  2.4  2.77 2.3 ]]
```

Mathematical Functions

In [ ]:
```
there several mathematical functions we can use in numpy:-
    1] Addition
    2] Multiplication
    3] Substraction
    4] Division
    5] Sum
    6] square
    7] squareroot
    8] cube
    9] cuberoot

```

In [153]:
```
a=np.random.randint(10,80,(4,4))
b=np.random.randint(10,80,(4,4))
print(a)
print(b)
```

```
[[49 12 68 44]
 [55 35 24 76]
 [37 14 73 46]
 [43 40 61 67]]
[[19 13 10 53]
 [66 43 76 45]
 [74 10 66 63]
 [40 16 19 34]]
```

In [155]:
```
#Addition
print(a+b)
print("*"*80)
print(np.add(a,b))
```

```
[[ 68  25  78  97]
 [121  78 100 121]
 [111  24 139 109]
 [ 83  56  80 101]]
********************************************************************************
[[ 68  25  78  97]
 [121  78 100 121]
 [111  24 139 109]
 [ 83  56  80 101]]
```

In [156]:
```
#Multiplication
print(a*b)
print("*"*80)
print(np.multiply(a,b))
```

```
[[ 931  156  680 2332]
 [3630 1505 1824 3420]
 [2738  140 4818 2898]
 [1720  640 1159 2278]]
********************************************************************************
[[ 931  156  680 2332]
 [3630 1505 1824 3420]
 [2738  140 4818 2898]
 [1720  640 1159 2278]]
```

In [157]:
```
#Substraction
print(a-b)
print("*"*80)
```

```
[[ 30  -1  58  -9]
 [-11  -8 -52  31]
 [-37   4   7 -17]
 [  3  24  42  33]]
********************************************************************************
```

In [158]:
```
#Division
print(a/b)
print("*"*80)
```

```
[[2.57894737 0.92307692 6.8        0.83018868]
 [0.83333333 0.81395349 0.31578947 1.68888889]
 [0.5        1.4        1.10606061 0.73015873]
 [1.075      2.5        3.21052632 1.97058824]]
********************************************************************************
```

sum()

In [159]:
```
a
```

Out[159]:
```
array([[49, 12, 68, 44],
       [55, 35, 24, 76],
       [37, 14, 73, 46],
       [43, 40, 61, 67]])
```

In [160]:
```
#getting sum column wise
np.sum(a,axis=0)
```

Out[160]:
```
array([184, 101, 226, 233])
```

In [161]:
```
#getting sum row wise
np.sum(a,axis=1)
```

Out[161]:
```
array([173, 190, 170, 211])
```

In [162]:
```
#getting sum of whole array
np.sum(a)
```

Out[162]:
```
744
```

np.sqrt() and square

In [ ]:
```
# np.sqrt :-It gives squareroot of the array
```

In [164]:
```
a

```

Out[164]:
```
array([[11, 53, 48, 30],
       [23, 38, 18, 45],
       [57, 19, 39, 48],
       [19, 44, 49, 55],
       [31, 35, 17, 29]])
```

In [166]:
```
print("Square of array is:-\n",a**2)
```

```
Square of array is:-
 [[ 121 2809 2304  900]
 [ 529 1444  324 2025]
 [3249  361 1521 2304]
 [ 361 1936 2401 3025]
 [ 961 1225  289  841]]
```

In [168]:
```
print("Squareroot of array is:-\n",np.sqrt(a).round(2))
```

```
Squareroot of array is:-
 [[3.32 7.28 6.93 5.48]
 [4.8  6.16 4.24 6.71]
 [7.55 4.36 6.24 6.93]
 [4.36 6.63 7.   7.42]
 [5.57 5.92 4.12 5.39]]
```

np.cbrt() and cube

In [169]:
```
a
```

Out[169]:
```
array([[11, 53, 48, 30],
       [23, 38, 18, 45],
       [57, 19, 39, 48],
       [19, 44, 49, 55],
       [31, 35, 17, 29]])
```

In [170]:
```
print("Cube of array is:-\n",a**3)
```

```
Cube of array is:-
 [[  1331 148877 110592  27000]
 [ 12167  54872   5832  91125]
 [185193   6859  59319 110592]
 [  6859  85184 117649 166375]
 [ 29791  42875   4913  24389]]
```

In [172]:
```
print("Cuberoot of array is:-\n",np.cbrt(a).round(2))
```

```
Cuberoot of array is:-
 [[2.22 3.76 3.63 3.11]
 [2.84 3.36 2.62 3.56]
 [3.85 2.67 3.39 3.63]
 [2.67 3.53 3.66 3.8 ]
 [3.14 3.27 2.57 3.07]]
```