'Numerical Python'

- In 2005, Travis Oliphant created NumPy package
- package for data analysis and scientific computing with Python
- The most important object defined in NumPy is an N-dimensional array type called **ndarray**
- Every item in an ndarray takes the **same size** of block in the memory
- Each element in ndarray is an **object** of data-type object (called dtype).
- How to install: **pip install numpy**

Import Numpy

```
import numpy as np
```

Creation of one dimension numpy array

```
array1=np.array([1,2,3,4])
print('One dimensional numpy array ', array1)
```

```
One dimensional numpy array  [1 2 3 4]
```

Creation of two dimension numpy array

```
array2=np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
print('Two dimensional numpy array \n', array2)
```

```
Two dimensional numpy array
 [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

Creation of 3-D Numpy array

```
array3=np.array([[[1,2,3,4], [5,6,7,8]],[[13,14,15,16],[17,18,19,20]]])
print('Three dimensional numpy array \n', array3)
```

```
Three dimensional numpy array
```

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

 [[13 14 15 16]
  [17 18 19 20]]]
```

checking CLASS of array

```
array1=np.array([1,2,3,4])
array2=np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
array3=np.array([[[1,2,3,4], [5,6,7,8]],[[13,14,15,16],[17,18,19,20]]])

print(' Class of  one dimensional numpy array ',type(array1))
print(' Class of  Two dimensional numpy array ',type(array2))
print(' Class of  three dimensional numpy array ',type(array3))

```

```
 Class of  one dimensional numpy array  <class 'numpy.ndarray'>
 Class of  Two dimensional numpy array  <class 'numpy.ndarray'>
 Class of  three dimensional numpy array  <class 'numpy.ndarray'>
```

Checking DATATYPE of numpy array

```
array1=np.array([1,2,3,4])
array2=np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
array3=np.array([[[1,2,3,4], [5,6,7,8]],[[13,14,15,16],[17,18,19,20]]])
print(' Data type one dimensional numpy array ',array1.dtype)
print(' Data type Two dimensional numpy array ',array2.dtype)
print(' Data type three dimensional numpy array ',array3.dtype)
```

```
 Data type one dimensional numpy array  int64
 Data type Two dimensional numpy array  int64
 Data type three dimensional numpy array  int64
```

Checking DIMENSION of numpy array

```
array1=np.array([1,2,3,4])
array2=np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
array3=np.array([[[1,2,3,4], [5,6,7,8]],[[13,14,15,16],[17,18,19,20]]])
print('dimension of one dimensional numpy array ',array1.ndim)
print('dimension of Two dimensional numpy array ',array2.ndim)
print('dimension of three dimensional numpy array ',array3.ndim)

```

```
dimension of one dimensional numpy array  1
dimension of Two dimensional numpy array  2
dimension of three dimensional numpy array  3
```

# Checking SHAPE of numpy array

```
array1=np.array([1,2,3,4])
array2=np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
array3=np.array([[[1,2,3,4], [5,6,7,8]],[[13,14,15,16],[17,18,19,20]]])
print('shape of one dimensional numpy array ',array1.shape)
print('shape of Two dimensional numpy array ',array2.shape)
print('shape of three dimensional numpy array ',array3.shape)

```

```
shape of one dimensional numpy array  (4,)
shape of Two dimensional numpy array  (3, 4)
shape of three dimensional numpy array  (2, 2, 4)
```

Checking NUMBER OF ELEMENTS of numpy array

```
array1=np.array([1,2,3,4])
array2=np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
array3=np.array([[[1,2,3,4], [5,6,7,8]],[[13,14,15,16],[17,18,19,20]]])
print('size of one dimensional numpy array ',array1.size)
print('size of Two dimensional numpy array ',array2.size)
print('size of three dimensional numpy array ',array3.size)
```

```
size of one dimensional numpy array  4
size of Two dimensional numpy array  12
size of three dimensional numpy array  16
```

** Number of element in i-1th dimension array**

```
array1=np.array([1,2,3,4])
array2=np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
array3=np.array([[[1,2,3,4], [5,6,7,8]],[[13,14,15,16],[17,18,19,20]]])
print(' number of element or Zeroth dimension in one dimensional numpy array ',
print(' number of  one dimensional numpy array ',len(array2))
print(' number of  two dimensional numpy array ',len(array3))
```

```
 number of element or Zeroth dimension in one dimensional numpy array  4
 number of  one dimensional numpy array  3
 number of  two dimensional numpy array  2
```

```
row,col=array2.shape
print(row,col)
a=(np.array([[1,2,3,4]])).ndim
```

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

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

Lenght of array in bytes

```
x = np.array([1,2,3,4,5])
print(x.itemsize)# size of each element in bytes
print((x.itemsize)*x.size)
print(x.nbytes)# size of complete array
```

```
8
40
40
```

conversion of LIST, TUPLE, DICTIONARY, SET to Numpy array

- np.array()
- np.asarray()

```
a=[1,2,3,4]#LIST
print(type(a))
b=np.array(a)
print(type(b))
print("\n")
print("\n")
a=(1,2,3,4)#TUPLE
print(type(a))
b=np.array(a)
print(type(b))
print("\n")
print("\n")

a={1,2,2,3,4}#SET
print(type(a))
b=np.array(a)
print(type(b))
print(a)
print(b)
print("\n")
print("\n")
a={'name':1,'address':2}#DICTIONARY
print(type(a))
b=np.array(a)
print(type(b))
```

```
<class 'list'>
<class 'numpy.ndarray'>




<class 'tuple'>
<class 'numpy.ndarray'>




<class 'set'>
<class 'numpy.ndarray'>
{1, 2, 3, 4}
{1, 2, 3, 4}




<class 'dict'>
<class 'numpy.ndarray'>
```

```
a=[1,2,3,4]#LIST
print(type(a))
a=np.asarray(a)
print(a.dtype)
```

```
<class 'list'>
int64
```

Indexing of NUMPY array

- same as accessing an array element.
- access an array element by referring to its index number.
- indexes in NumPy arrays start with 0

```
array1=np.array([1,2,3,4])
print('first element of array1:',array1[0],'\n','second element of array1:',arra
      'third element of array1',array1[2],'\n','fourth element of array1:',array
```

```
first element of array1: 1
 second element of array1: 2
 third element of array1 3
 fourth element of array1: 4
```

```
array2=np.array([[1,2],[5,6]])
print('first element of array2:',array2[0][0],'\n','second element of array2:',a
      'third element of array2',array2[1][0],'\n','fourth element of array2:',a


print(array2[0])
print(array2[1])
###########Print Class
print(type(array2[0]))
print(type(array2[1][1]))

```

```
first element of array2: 1
 second element of array2: 2
 third element of array2 5
 fourth element of array2: 6
[1 2]
[5 6]
<class 'numpy.ndarray'>
<class 'numpy.int64'>
```

```

array3=np.array([[[1,2,3,4], [5,6,7,8]],[[13,14,15,16],[17,18,19,20]]])
print('first element of array3:',array3[0][0][0],'\n')


print(array3[0][0])
print("\n")
print(array3[1][1])
print("\n")
print(array3[0])
print("\n")
print(array3[1])
############################Print class
print(type(array3[0][0]))
print(type(array3[1][1]))
print(type(array3[0]))
print(type(array3[1]))
```

```
first element of array3: 1

[1 2 3 4]


[17 18 19 20]


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


[[13 14 15 16]
 [17 18 19 20]]
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
```

```
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
```

NumPy Array Slicing

- Slicing: taking elements from one given index to another given index.
- pass slice: **[start:end] or [start:end:step]**.
- start =0 (by default)
- end = length of array in that dimension(by default)
- step = 1 (by default)

```
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[4:])
print(arr[:4])
```

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

```
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[-3:-1])# Use the minus operator to refer to an index from the end

print(arr[1:5:2])# Return every other element from index 1 to index 5

print(arr[::2])# Return every other element from the entire array:
```

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

```
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr[1,4])
print('\n')
print(arr[1, 1:4])
print("\n")
print(arr[0:2, 2])
print("\n")
#print(arr[0:4, 1])
#print("\n")
print(arr[0:2, 1:4])
```

```
10


[7 8 9]


[3 8]
```

```
[[2 3 4]
 [7 8 9]]
```

DATA TYPES

strings - used to represent text data, the text is given under quote marks. e.g. "ABCD" integer - used to represent integer numbers. e.g. -1, -2, -3 float - used to represent real numbers. e.g. 1.2, 42.42 boolean - used to represent True or False. complex - used to represent complex numbers. e.g. 1.0 + 2.0j, 1.5 + 2.5j

```
a=np.array([1,2,3,4])
b=np.array([True,False])
c=np.array(['this','is','my','name'])
d=np.array([2.0,3.6,6.77])
e=np.array([3+4j])
print(a.dtype)
print(b.dtype)
print(c.dtype)
print(d.dtype)
print(e.dtype)

```

```
int64
bool
<U4
float64
complex128
```

Creating Arrays With a Defined Data Type

i - integer b - boolean f - float O - object S - string

```
a=np.array([1,2,3,4],dtype='f')
print(a)
a=np.array([1,2,3,4],dtype='O')
print(a)
a=np.array([1,2,3,4],dtype='b')
print(a)
a=np.array([1,2,3,4],dtype='S')
print(a)

```

```
[1. 2. 3. 4.]
[1 2 3 4]
[1 2 3 4]
[b'1' b'2' b'3' b'4']
```

```
#anotherway of defining datatype
a=np.array([1,2,3,4])
print(a.dtype)
b=np.array([1,2,3,4],dtype=np.int8)
print(b.dtype)
c=a+b
print(c)
print(c.dtype)

```

```
int64
int8
[2 4 6 8]
int64
```

```
b=np.array([1.2,2,3,4])
print(b)
print(b.dtype)
c=np.array([1.2,2,3,4],np.float32)
print(c.dtype)
```

```
[1.2 2.  3.  4. ]
float64
float32
```

SHAPE and RESHAPE

- ndmin
- reshape

```
import numpy as np
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
print(arr.shape)
```

```
(2, 4)
```

```

arr = np.array([1, 2, 3, 4], ndmin=1)
print(arr)
print('shape of array :', arr.shape)
print("\n")
arr = np.array([1, 2, 3, 4], ndmin=2)
print(arr)
print('shape of array :', arr.shape)
print("\n")
arr = np.array([1, 2, 3, 4], ndmin=4)
print(arr)
print('shape of array :', arr.shape)

```

```
[1 2 3 4]
shape of array : (4,)


[[1 2 3 4]]
shape of array : (1, 4)


[[[[1 2 3 4]]]]
shape of array : (1, 1, 1, 4)
```

```
a=np.array([[[1,2],[3,4]]])
print(a.shape,a[0][0][0])
a=np.array([[1,2],[3,4]],ndmin=3)
print(a.shape)
a=np.array([[1,2],[3,4]],ndmin=4)
print(a.shape)
a=np.array([[1,2],[3,4]],ndmin=1)
print(a.shape,a[1])
```

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

```
#reshape
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
newarr = arr.reshape(4, 3)
print(newarr)
```

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

```
newarr = arr.reshape(3, 4)
print(newarr)
print(newarr.shape)
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
(3, 4)
```

```
newarr = np.array(arr.reshape(3, 4),ndmin=4)
print(newarr)
print(newarr.shape)
```

```
[[[[ 1  2  3  4]
   [ 5  6  7  8]
   [ 9 10 11 12]]]]
(1, 1, 3, 4)
```

Creation of EMPTY,ZEROS,ONES numpyarray

- numpy.empty(shape, dtype=float, order='C/F')
- numpy.zeros(shape, dtype=float, order='C/F')
- numpy.ones(shape, dtype=None, order='C/F')

```
x = np.empty([3,2])
print(x)
```

```
[[4.9e-324 2.0e-323]
 [9.9e-324 2.5e-323]
 [1.5e-323 3.0e-323]]
```

```
x = np.empty([3,2],dtype=np.uint8)
print(x)
```

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

```
x = np.empty([3,2],dtype=np.float32)
print(x)
```

```
[[1.e-45 0.e+00]
 [0.e+00 0.e+00]
 [3.e-45 0.e+00]]
```

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

```
[[0. 0.]
 [0. 0.]
 [0. 0.]] float64
```

```
x=np.zeros([3,2],np.uint8)
print(x,x.dtype)
```

```
[[0 0]
 [0 0]
 [0 0]] uint8
```

```
x=np.ones([3,2])
print(x,x.dtype)
```

```
[[1. 1.]
 [1. 1.]
 [1. 1.]] float64
```

```
x=np.zeros([3,2],np.int64)
print(x,x.dtype)
```

```
[[0 0]
 [0 0]
 [0 0]] int64
```

JOIN

- join arrays by axes.
- axis is not explicitly passed, it is taken as 0.

- axix=1 (rowwise)
- axis=0 (columnwise)

- np.concatenate(), np.stack(), np.dstack()

```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr = np.concatenate((arr1, arr2),axis=0)
print(arr)
```

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

```
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6], [7, 8]])
arr = np.concatenate((arr1, arr2), axis=1)
print(arr)
```

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

```
arr = np.concatenate((arr1, arr2), axis=0)
print(arr)
```

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

Joining Arrays Using Stack Functions

```
arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

arr = np.stack((arr1, arr2), axis=1)

```

```
print(arr)
print(arr.shape)
```

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

```

arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

arr = np.hstack((arr1, arr2))

print(arr)
```

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

```
arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

arr = np.vstack((arr1, arr2))

print(arr)
```

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

```
arr1 = np.array([1, 2, 3,7])

arr2 = np.array([4, 5, 6,8])

arr = np.dstack((arr1, arr2))

print(arr)
```

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

NumPy Splitting Array

np.array_split

```
arr = np.array([1, 2, 3, 4, 5, 6])
newarr = np.array_split(arr, 3)
print(newarr)
#print(newarr.dtype)# AttributeError: 'list' object has no attribute 'dtype'
```

```
newarr=np.array(newarr)
print(type(newarr))
```

```
[array([1, 2]), array([3, 4]), array([5, 6])]
<class 'numpy.ndarray'>
```

```
arr = np.array([1, 2, 3, 4, 5, 6,7])
newarr = np.array_split(arr, 3)
print(newarr)
#print(newarr.dtype)# AttributeError: 'list' object has no attribute 'dtype'
newarr=np.array(newarr)
print(type(newarr))
```

```
[array([1, 2, 3]), array([4, 5]), array([6, 7])]
<class 'numpy.ndarray'>
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: VisibleDeprec
  """
```

```
arr = np.array([1, 2, 3, 4, 5, 6])
newarr = np.split(arr, 3)
print(newarr)
#print(newarr.dtype)# AttributeError: 'list' object has no attribute 'dtype'
newarr=np.array(newarr)
print(type(newarr))
```

```
[array([1, 2]), array([3, 4]), array([5, 6])]
<class 'numpy.ndarray'>
```

```
arr = np.array([1, 2, 3, 4, 5, 6,7])#array split does not result in an equal di
#newarr = np.split(arr, 3)
print(newarr)
#print(newarr.dtype)# AttributeError: 'list' object has no attribute 'dtype'
newarr=np.array(newarr)
print(type(newarr))
```

```
[[1 2]
 [3 4]
 [5 6]]
<class 'numpy.ndarray'>
```

```
arr = np.array([[1, 2, 3], [4, 5, 6]])
newarr = np.array_split(arr, 3)
print(newarr)
#print(newarr.dtype)# AttributeError: 'list' object has no attribute 'dtype'
newarr=np.array(newarr)
print(type(newarr))
print(newarr[0])
```

```
[array([[1, 2, 3]]), array([[4, 5, 6]]), array([], shape=(0, 3), dtype=int64)
<class 'numpy.ndarray'>
[[1 2 3]]
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: VisibleDeprec
  """
```

```
arr = np.array([[1, 2, 3], [4, 5, 6]])
newarr = np.array_split(arr, 3,axis=1)
print(newarr)
#print(newarr.dtype)# AttributeError: 'list' object has no attribute 'dtype'
newarr=np.array(newarr)
print(type(newarr))
print(type(newarr))
```

```
[array([[1],
       [4]]), array([[2],
       [5]]), array([[3],
       [6]])]
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
```

Sorting

np.sort

```
arr = np.array([3, 2, 0, 1])
print(np.sort(arr))
```

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

```
arr = np.array(['banana', 'cherry', 'apple'])
print(np.sort(arr))
```

```
['apple' 'banana' 'cherry']
```

```
import numpy as np

arr = np.array([[3, 2, 4], [5, 0, 1]])

print(np.sort(arr))
```

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

Searching Arrays

np.where(): return the indexes that get a match

```
import numpy as np

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

```
x = np.where(arr%2 == 1)
print(x)
```

ARRAY FROM NUMERICAL RANGES

- numpy.arange(start, stop, step, dtype)
- numpy.linspace(start, stop, num, endpoint, retstep, dtype)
- numpy.logspace(start, stop, num, endpoint, base, dtype)

```
x = np.arange(5)
print(x)
```

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

```
x = np.arange(5, dtype=float)
print (x)
```

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

```
x = np.arange(10,20,2,dtype=float)
print(x)
```

```
[10. 12. 14. 16. 18.]
```

```
x = np.linspace(5,10)#evenlyspaced samples, by default 10 samples
print(x)
```

```
[ 5.          5.10204082  5.20408163  5.30612245  5.40816327  5.51020408
  5.6122449   5.71428571  5.81632653  5.91836735  6.02040816  6.12244898
  6.2244898   6.32653061  6.42857143  6.53061224  6.63265306  6.73469388
  6.83673469  6.93877551  7.04081633  7.14285714  7.24489796  7.34693878
  7.44897959  7.55102041  7.65306122  7.75510204  7.85714286  7.95918367
  8.06122449  8.16326531  8.26530612  8.36734694  8.46938776  8.57142857
  8.67346939  8.7755102   8.87755102  8.97959184  9.08163265  9.18367347
  9.28571429  9.3877551   9.48979592  9.59183673  9.69387755  9.79591837
  9.89795918 10.        ]
```

```
x = np.linspace(5,10,20)
print(x)
```

```
[ 5.          5.26315789  5.52631579  5.78947368  6.05263158  6.31578947
  6.57894737  6.84210526  7.10526316  7.36842105  7.63157895  7.89473684
  8.15789474  8.42105263  8.68421053  8.94736842  9.21052632  9.47368421
  9.73684211 10.        ]
```

```
x = np.linspace(5,10,20)
print(x)
```

```
[ 5.          5.26315789  5.52631579  5.78947368  6.05263158  6.31578947
```

```
 6.57894737  6.84210526  7.10526316  7.36842105  7.63157895  7.89473684
 8.15789474  8.42105263  8.68421053  8.94736842  9.21052632  9.47368421
 9.73684211 10.        ]
```

```
a = np.logspace(1,10,num = 10, base = 2)
print (a)
```

```
[   2.    4.    8.   16.   32.   64.  128.  256.  512. 1024.]
```

STATISTICAL FUNCTIONS

Finding MIN,MAX,UNIQUE,ALL,ANY

```
a=[0,1,2,2,3,4]
print(np.max(a))
print(np.min(a))
print(np.unique(a))
print(np.all(a))
print(np.any(a))

```

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

```
import numpy as np
a = np.array([[3,7,5],[8,4,3],[2,4,9]])
print (a )

```

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

```
print (np.amin(a,1) )

print (np.amin(a,0))

print (np.amax(a) )

print (np.amax(a, axis = 0))
```

```
[3 3 2]
[2 4 3]
9
[8 7 9]
```

Mean,median,mode,Variance,standard deviation

```
a=[[1,2],[3,4]]
print(np.mean(a))
```

```
2.5
```

```

```

```
b=np.mean(a,axis=0)
c=np.mean(a,axis=1)

print('columnwise_mean',b)
print('rowwise_mean',c)
```

```
columnwise_mean [2. 3.]
rowwise_mean [1.5 3.5]
```

```
print(np.var(a))
b=np.var(a,axis=0)
c=np.var(a,axis=1)

print('columnwise_variance',b)
print('rowwise_variance',c)
```

```
1.25
columnwise_variance [1. 1.]
rowwise_variance [0.25 0.25]
```

```
d=np.std(a)
print(d)
```

```
1.118033988749895
```

```
a= [1,1,2,3,4,6,18,20,]
median = np.median(a)
print(median)
```

```
3.5
```

```
#returns the range (maximum-minimum)
a = np.array([[3,7,5],[8,4,3],[2,4,9]])

print (a )
print ('\n'  )

print (np.ptp(a))
print ('\n' )

print ('Applying ptp() function along axis 1:')
print (np.ptp(a, axis = 1) )
print ('\n')

print ('Applying ptp() function along axis 0:')
print (np.ptp(a, axis = 0) )
```

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


7


Applying ptp() function along axis 1:
[4 5 7]


Applying ptp() function along axis 0:
[6 3 6]
```

numpy random

```
#Generate a random integer from 0 to 100:
from numpy import random
x = random.randint(100)
print(x)
```

```
22
```

```
#Generate a random float from 0 to 1:
x = random.rand()
print(x)
```

```
0.6612179785899164
```

```
x=random.randint(100, size=(5))
print(x)
```

```
[23 76 39 43 22]
```

```
x=random.randint(100, size=(10))
print(x)
```

```
[26 63 44 92 79 30  4 46 69 48]
```

```
x = random.randint(100, size=(3, 5))
print(x)
```

```
[[31 43 29 21 98]
 [48 93 65 34 98]
 [74 68 17 48 16]]
```

```
x = random.rand(3, 5)
print(x)
```

```
[[0.19129665 0.78706377 0.37950255 0.26225171 0.73637203]
 [0.18175589 0.4952115  0.9233343  0.7763049  0.20973812]
 [0.90227649 0.69502188 0.24565797 0.8224613  0.2659112 ]]
```

```
x = random.choice([3, 5, 7, 9])
print(x)
```

```
7
```

```
x = random.choice([3, 5, 7, 9], size=(3, 5))

print(x)
```

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

String operations

- **numpy.char.add()**: elementwise string concatenation.
- **numpy.char.multiply()**: multiple concatenation.
- **np.char.center**: returns an array of the required width so that the input string is centered and padded on the left and right with fillchar
- **numpy.char.capitalize()**: First letter capital
- **numpy.char.title()**: first letter of each word capitalized
- **numpy.char.lower()**: returns an array with elements converted to lowercase
- **numpy.char.upper()**: return the uppercase array elements
- **numpy.char.split()**: This function returns a list of words in the input string. By default, a whitespace is used as a separator. Otherwise the specified separator character is used to spilt the string.
- **numpy.char.strip()**: stripped of the specified characters leading and/or trailing
- **np.char.join**:
- **np.char.replace**:
- **np.char.encode/decode**:

```
print (np.char.add(['hello'],[' xyz']) )
print ('\n')
print (np.char.add(['hello', 'hi'],[' abc', ' xyz']))
```

```
['hello xyz']
```

```
['hello abc' 'hi xyz']
```

```
print (np.char.multiply('Hello ',3))
```

```
Hello Hello Hello
```

```

print (np.char.center('hello', 20,fillchar = '*'))
```

```
*******hello********
```

```
a= np.char.capitalize('hello world')
print(a)
```

```
Hello world
```

```

a=np.char.title('my name is ramesh?')
print(a)
```

```
My Name Is Ramesh?
```

```
a= np.char.lower(['HELLO','WORLD'])
print(a)
```

```
['hello' 'world']
```

```
a= np.char.upper(['hello','world'])
print(a)
```

```
['HELLO' 'WORLD']
```

```
#This function returns a list of words in the input string. By default, a white
#is used as a separator. Otherwise the specified separator character is used to
print (np.char.split ('hello how are you?') )
print (np.char.split ('TutorialsPoint,Hyderabad,Telangana', sep = ','))
```

```
['hello', 'how', 'are', 'you?']
['TutorialsPoint', 'Hyderabad', 'Telangana']
```

```
#  returns a copy of array with elements stripped of
#the specified characters leading and/or trailing in it.
print (np.char.strip('ashok arora','a') )
print (np.char.strip(['arora','admin','java'],'a'))
```

```
shok aror
['ror' 'dmin' 'jav']
```

```
#returns a string in which the individual characters are joined by separator cha
a= np.char.join(':','dmy')
print(a)
b= np.char.join([':','-'],['dmy','ymd'])
print(b)
```

```
d:m:y
['d:m:y' 'y-m-d']
```

```
a= np.char.replace ('He is a good boy', 'is', 'was')
print(a)
```

```
He was a good boy
```

```
###code decode
a = np.char.encode('hello', 'cp500')
print (a )
print (np.char.decode(a,'cp500'))
```

```
b'\x88\x85\x93\x93\x96'
hello
```

ARITHMETIC OPERATIONS

```
import numpy as np
a = np.arange(9, dtype = np.float_).reshape(3,3)

print('First array:')
print (a)
print ('\n')

print ('Second array:' )
b = np.array([10,10,10])
print (b )
print ('\n' )

print ('Add the two arrays:')
print (np.add(a,b) )
print ('\n'  )

print ('Subtract the two arrays:')
print (np.subtract(a,b) )
print ('\n'  )

print ('Multiply the two arrays:' )
print (np.multiply(a,b) )
print ('\n' )

print ('Divide the two arrays:' )
print (np.divide(a,b))
```

```
First array:
```

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


Second array:
[10 10 10]


Add the two arrays:
[[10. 11. 12.]
 [13. 14. 15.]
 [16. 17. 18.]]


Subtract the two arrays:
[[-10.  -9.  -8.]
 [ -7.  -6.  -5.]
 [ -4.  -3.  -2.]]


Multiply the two arrays:
[[ 0. 10. 20.]
 [30. 40. 50.]
 [60. 70. 80.]]


Divide the two arrays:
[[0.  0.1 0.2]
 [0.3 0.4 0.5]
 [0.6 0.7 0.8]]
```

```
a = np.array([0.25, 1.33, 1, 0, 100])
print (np.reciprocal(a) )
```

```
[4.        0.7518797 1.              inf 0.01     ]
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: RuntimeWarnin
```

```
a = np.array([10,100,1000])
print (a)
print (np.power(a,2) )
```

```
[  10  100 1000]
[    100   10000 1000000]
```

```
a = np.array([10,20,30])
b = np.array([3,5,7])
c=np.mod(a,b)
print(c)
d=np.remainder(a,b)
print(d)
```

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

```
##########complex number
a = np.array([-5.6j, 0.2j, 11. , 1+1j])

print ('Our array is:' )
print (a)
print ('\n')
print ('Applying real() function:',np.real(a),'\n' )
print ('Applying imag() function:',np.imag(a), '\n')
print ('Applying conj() function:' ,np.conj(a) , '\n' )
print ('Applying angle() function:' ,np.angle(a) ,'\n' )
print ('Applying angle() function again (result in degrees)' )
print (np.angle(a, deg = True))
```

```
Our array is:
[-0.-5.6j  0.+0.2j 11.+0.j   1.+1.j ]


Applying real() function: [-0.  0. 11.  1.]

Applying imag() function: [-5.6  0.2  0.   1. ]

Applying conj() function: [-0.+5.6j  0.-0.2j 11.-0.j   1.-1.j ]

Applying angle() function: [-1.57079633  1.57079633  0.          0.78539816]

Applying angle() function again (result in degrees)
[-90.  90.   0.  45.]
```

✓ 0s completed at 9:19 AM