NUMPY

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

Create an arry hich contain first 19 digit

In [ ]:
```
arr=np.arange(20)
```

In [ ]:
```
arr
```

Out[ ]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19])
```

In [ ]:
```
arr_1d=np.array([1,2,3,4])
print(arr_1d)
```

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

In [ ]:
```
type(arr_1d)
```

Out[ ]:
```
numpy.ndarray
```

In [ ]:
```
arr_1d.ndim
```

Out[ ]:
```
1
```

In [ ]:
```
arr_2d=np.array([[1,2,3,4], [5,6,7,8,]])
print(arr_2d)
```

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

Array Dimension type

In [ ]:
```
arr_2d.ndim
```

Out[ ]:
```
2
```

Array Size

In [ ]:
```
arr_1d.size
```

Out[ ]:
```
4
```

In [ ]:
```
arr_2d.size
```

Out[ ]:
```
8
```

Array Shape which tells us Rows & Columns

we 2 rows & 4 Columns

In [ ]:
```
arr_2d.shape
```

Out[ ]:
```
(2, 4)
```

dtype(int 32) tells us about data type which is int and saved in 32 bits

In [ ]:
```
arr_2d.dtype
```

Out[ ]:
```
dtype('int32')
```

In [ ]:
```
arr_2d
```

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

In [ ]:
```
arr_3d=np.array([[1,1,1], [1,1,1],[1,0,1]])
arr_3d
```

Out[ ]:
```
array([[1, 1, 1],
       [1, 1, 1],
       [1, 0, 1]])
```

In [ ]:
```
mx_1s =np.ones(5)
mx_1s
```

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

In [ ]:
```
mx_1s.dtype
```

Out[ ]:
```
dtype('float64')
```

In [ ]:
```
mx_2s = np.ones((3,4))
print(mx_2s)
```

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

In [ ]:
```
mx_1s = np.ones((3,4), dtype = int )
mx_1s
```

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

In [ ]:
```
mx_0s=np.zeros((4,6),)
mx_0s
```

Out[ ]:
```
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.]])
```

In [ ]:
```
mx_0s = np.zeros((4,6), dtype= bool)
mx_0s
```

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

In [ ]:
```
mx_0s = np.zeros((4,6), dtype= str)
mx_0s
```

Out[ ]:
```
array([['', '', '', '', '', ''],
       ['', '', '', '', '', ''],
       ['', '', '', '', '', ''],
       ['', '', '', '', '', '']], dtype='<U1')
```

In [ ]:
```
mx_0s = np.empty((3,3))
mx_0s
```

Out[ ]:
```
array([[0.00000000e+000, 0.00000000e+000, 0.00000000e+000],
       [0.00000000e+000, 0.00000000e+000, 5.79044937e-321],
       [6.23041391e-307, 1.60219034e-306, 1.89145199e-307]])
```

In [ ]:
```

```

Python Numpy Array Slicing

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

In [ ]:
```
mx = np.arange(1,101).reshape(10,10)
mx
```

Out[ ]:
```
array([[  1,  2,  3,  4,  5,  6,  7,  8,  9, 10],
       [ 11, 12, 13, 14, 15, 16, 17, 18, 19, 20],
       [ 21, 22, 23, 24, 25, 26, 27, 28, 29, 30],
       [ 31, 32, 33, 34, 35, 36, 37, 38, 39, 40],
       [ 41, 42, 43, 44, 45, 46, 47, 48, 49, 50],
       [ 51, 52, 53, 54, 55, 56, 57, 58, 59, 60],
       [ 61, 62, 63, 64, 65, 66, 67, 68, 69, 70],
       [ 71, 72, 73, 74, 75, 76, 77, 78, 79, 80],
       [ 81, 82, 83, 84, 85, 86, 87, 88, 89, 90],
       [ 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]])
```

In [ ]:
```
## Access specific value from matrix using index
mx[0,0]
```

Out[ ]:
```
1
```

In [ ]:
```
## Dimension type
mx[0,0].ndim
```

Out[ ]:
```
0
```

In [ ]:
```
## Print specific row or column
mx[0]
```

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

In [ ]:
```
mx[:,0]
```

Out[ ]:
```
array([ 1, 11, 21, 31, 41, 51, 61, 71, 81, 91])
```

In [ ]:
```
mx[:,0:1]
```

Out[ ]:
```
array([[ 1],
       [11],
       [21],
       [31],
       [41],
       [51],
       [61],
       [71],
       [81],
       [91]])
```

In [ ]:
```
mx[:,0:1].ndim
```

Out[ ]:
```
2
```

In [ ]:
```
mx
```

Out[ ]:
```
array([[  1,  2,  3,  4,  5,  6,  7,  8,  9, 10],
       [ 11, 12, 13, 14, 15, 16, 17, 18, 19, 20],
       [ 21, 22, 23, 24, 25, 26, 27, 28, 29, 30],
       [ 31, 32, 33, 34, 35, 36, 37, 38, 39, 40],
       [ 41, 42, 43, 44, 45, 46, 47, 48, 49, 50],
       [ 51, 52, 53, 54, 55, 56, 57, 58, 59, 60],
       [ 61, 62, 63, 64, 65, 66, 67, 68, 69, 70],
       [ 71, 72, 73, 74, 75, 76, 77, 78, 79, 80],
       [ 81, 82, 83, 84, 85, 86, 87, 88, 89, 90],
       [ 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]])
```

In [ ]:
```
mx[1,4:3,4]
```

Out[ ]:
```
array([[12, 13, 14],
       [22, 23, 24],
       [32, 33, 34]])
```

In [ ]:
```
mx[1,4:1,4].ndim
```

Out[ ]:
```
2
```

In [ ]:
```
mx[:,2:4]
```

Out[ ]:
```
array([[  3,  4],
       [ 13, 14],
       [ 23, 24],
       [ 33, 34],
       [ 43, 44],
       [ 53, 54],
       [ 63, 64],
       [ 73, 74],
       [ 83, 84],
       [ 93, 94]])
```

In [ ]:
```
## size in byte
mx.itemsize
```

Out[ ]:
```
4
```

In [ ]:
```
mx.dtype
```

Out[ ]:
```
dtype('int32')
```

Python Numpy Conctination and Split

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

In [ ]:
```
m_1 = np.arange(1,17).reshape(4,4)
m_2 = np.arange(17,33).reshape(4,4)
```

In [ ]:
```
print(m_1, m_2)
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]] [[17 18 19 20]
 [21 22 23 24]
 [25 26 27 28]
 [29 30 31 32]]
```

In [ ]:
```
## conccination of list VS Array
list1=[2,4,6,8]
list2=[7,9,10]
list1 + list2
```

Out[ ]:
```
[2, 4, 6, 8, 7, 8, 9, 10]
```

In [ ]:
```
## See the Difference
m_1 + m_2
```

Out[ ]:
```
array([[18, 20, 22, 24],
       [26, 28, 30, 32],
       [34, 36, 38, 40],
       [42, 44, 46, 48]])
```

In [ ]:
```
np.concatenate((m_1, m_2))
```

Out[ ]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [13, 14, 15, 16],
       [17, 18, 19, 20],
       [21, 22, 23, 24],
       [25, 26, 27, 28],
       [29, 30, 31, 32]])
```

In [ ]:
```
## Column vise Concat
np.concatenate((m_1,m_2), axis= 1 )
```

Out[ ]:
```
array([[ 1,  2,  3,  4, 17, 18, 19, 20],
       [ 5,  6,  7,  8, 21, 22, 23, 24],
       [ 9, 10, 11, 12, 25, 26, 27, 28],
       [13, 14, 15, 16, 29, 30, 31, 32]])
```

In [ ]:
```
## Row vise Concat or virtival concatenate
np.vstack((m_1,m_2))
```

Out[ ]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [13, 14, 15, 16],
       [17, 18, 19, 20],
       [21, 22, 23, 24],
       [25, 26, 27, 28],
       [29, 30, 31, 32]])
```

In [ ]:
```
np.hstack((m_1,m_2))
```

Out[ ]:
```
array([[ 1,  2,  3,  4, 17, 18, 19, 20],
       [ 5,  6,  7,  8, 21, 22, 23, 24],
       [ 9, 10, 11, 12, 25, 26, 27, 28],
       [13, 14, 15, 16, 29, 30, 31, 32]])
```

In [ ]:
```
## Multiple array concatination
np.hstack((m_1,m_2,m_1))
```

Out[ ]:
```
array([[ 1,  2,  3,  4, 17, 18, 19, 20,  1,  2,  3,  4],
       [ 5,  6,  7,  8, 21, 22, 23, 24,  5,  6,  7,  8],
       [ 9, 10, 11, 12, 25, 26, 27, 28,  9, 10, 11, 12],
       [13, 14, 15, 16, 29, 30, 31, 32, 13, 14, 15, 16]])
```

Split Array

In [ ]:
```
np.split(m_1,2)
#see difference
```

Out[ ]:
```
[array([[1, 2, 3, 4],
        [5, 6, 7, 8]]),
 array([[ 9, 10, 11, 12],
        [13, 14, 15, 16]])]
```

In [ ]:
```
## Original Array
m_1
```

Out[ ]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [13, 14, 15, 16]])
```

In [ ]:
```
list1 = np.split(m_1,2)
type(list1)
```

Out[ ]:
```
list
```

In [ ]:
```
list1[0]
```

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

In [ ]:
```
type(list1[0])
```

Out[ ]:
```
numpy.ndarray
```

In [ ]:
```
m_1
```

Out[ ]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12],
       [13, 14, 15, 16]])
```

In [ ]:
```
## Column vise Split
np.split(m_1, 3, axis= 1)
```

Out[ ]:
```
[array([[ 1,  2],
        [ 5,  6],
        [ 9, 10],
        [13, 14]]),
 array([[ 3,  4],
        [ 7,  8],
        [11, 12],
        [15, 16]])]
```

In [ ]:
```
d1 = np.array([1,2,2,4,4])
```

In [ ]:
```
## [:,3] DESCRIBE ABOUT PICK VALUE FOR ARRAY 1 BEFORE 1ST INDEX & AFTER THAT PICK VALUE BEFORE 3 INDEX
np.split(d1,[1,3])
```

Out[ ]:
```
[array([1]), array([2, 2]), array([4, 4])]
```

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

In [ ]:
```
a=np.arange(1,101)
a
```

Out[ ]:
```
array([  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13,
        14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26,
        27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
        40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52,
        53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65,
        66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78,
        79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91,
        92, 93, 94, 95, 96, 97, 98, 99, 100])
```

In [ ]:
```
a=a.reshape(10,10)
```

In [ ]:
```
a.ndim
```

Out[ ]:
```
2
```

In [ ]:
```
a
```

Out[ ]:
```
array([[  1,  2,  3,  4,  5,  6,  7,  8,  9, 10],
       [ 11, 12, 13, 14, 15, 16, 17, 18, 19, 20],
       [ 21, 22, 23, 24, 25, 26, 27, 28, 29, 30],
       [ 31, 32, 33, 34, 35, 36, 37, 38, 39, 40],
       [ 41, 42, 43, 44, 45, 46, 47, 48, 49, 50],
       [ 51, 52, 53, 54, 55, 56, 57, 58, 59, 60],
       [ 61, 62, 63, 64, 65, 66, 67, 68, 69, 70],
       [ 71, 72, 73, 74, 75, 76, 77, 78, 79, 80],
       [ 81, 82, 83, 84, 85, 86, 87, 88, 89, 90],
       [ 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]])
```

In [ ]:
```
## b=np.array(a[1:4,1:4])
# b
```

In [ ]:
```
b=a[1:4,1:4]
b
```

Out[ ]:
```
array([[12, 13, 14],
       [22, 23, 24],
       [32, 33, 34]])
```

In [ ]:
```
c=b*5
c
```

Out[ ]:
```
array([[ 60,  65,  70],
       [110, 115, 120],
       [160, 165, 170]])
```

In [ ]:
```
a
```

Out[ ]:
```
array([[ 575,  600],
       [ 825, 4250]])
```

In [ ]:
```
a[1:4,1:4]=a[1:4,1:4]*5
a
```

Out[ ]:
```
array([[  1,  2,  3,  4,  5,  6,  7,  8,  9, 10],
       [ 11,  60,  65,  70,  15,  16,  17,  18,  19,  20],
       [ 21, 110, 115, 120,  25,  26,  27,  28,  29,  30],
       [ 31, 160, 165, 170,  35,  36,  37,  38,  39,  40],
       [ 41,  42,  43,  44,  45,  46,  47,  48,  49,  50],
       [ 51,  52,  53,  54,  55,  56,  57,  58,  59,  60],
       [ 61,  62,  63,  64,  65,  66,  67,  68,  69,  70],
       [ 71,  72,  73,  74,  75,  76,  77,  78,  79,  80],
       [ 81,  82,  83,  84,  85,  86,  87,  88,  89,  90],
       [ 91,  92,  93,  94,  95,  96,  97,  98,  99, 100]])
```

In [ ]:
```
a[1:4,1:4]=0
b=a.mean()
b
```

Out[ ]:
```
49.43
```

In [ ]:
```
a[1:4,1:4]=b
a
```

Out[ ]:
```
array([[  1,  2,  3,  4,  5,  6,  7,  8,  9, 10],
       [ 11,  49,  49,  49,  15,  16,  17,  18,  19,  20],
       [ 21,  49,  49,  49,  25,  26,  27,  28,  29,  30],
       [ 31,  49,  49,  49,  35,  36,  37,  38,  39,  40],
       [ 41,  42,  43,  44,  45,  46,  47,  48,  49,  50],
       [ 51,  52,  53,  54,  55,  56,  57,  58,  59,  60],
       [ 61,  62,  63,  64,  65,  66,  67,  68,  69,  70],
       [ 71,  72,  73,  74,  75,  76,  77,  78,  79,  80],
       [ 81,  82,  83,  84,  85,  86,  87,  88,  89,  90],
       [ 91,  92,  93,  94,  95,  96,  97,  98,  99, 100]])
```

In [ ]:
```
a.max()
```

Out[ ]:
```
100
```

In [ ]:
```
a.min()
```

Out[ ]:
```
1
```

In [ ]:
```
a.put
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
C:\Users\MN295-5.TAN\AppData\Local\Temp/ipykernel_2620/3378500945.py in <module>
----> 1 a.put

NameError: name 'a' is not defined
```

In [ ]:
```

```

NumPy Functions :

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

arange()

In [ ]:
```
#np.arange(START, END, STEP)
ar_1d = np.arange(1, 25, 2)
print(ar_1d)
```

```
[ 1  3  5  7  9 11 13 15 17 19 21 23]
```

In [ ]:
```
even_ar = np.arange(1, 13, 2)
even_ar
```

Out[ ]:
```
array([ 1,  3,  5,  7,  9, 11])
```

linspace()

In [ ]:
```
## we use linspace function when neeed values with equal gap
#np.linspace(start, end, how many values we need in start-end range)
np.linspace(1, 10, num=10)
```

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

reshape()

In [ ]:
```
## when we need to convert 1D arary into 2-3D array
```

In [ ]:
```
ar_2d = ar_1d.reshape(3,4)
ar_2d
```

Out[ ]:
```
array([[ 1,  3,  5,  7],
       [ 9, 11, 13, 15],
       [17, 19, 21, 23]])
```

In [ ]:
```
## conversion of 1D to 3D
#Array name.reshape(3D,Rows,Column)
ar_3d = ar_1d.reshape(2,3,2)
ar_3d
```

Out[ ]:
```
array([[[ 1,  3],
        [ 5,  7],
        [ 9, 11]],

       [[13, 15],
        [17, 19],
        [21, 23]]])
```

In [ ]:
```
arr=np.arange(1,13).reshape(4,3)
print(ar)
```

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

Ravel()

In [ ]:
```
#Ravel Function convert Multi-D to 1D
```

In [ ]:
```
ar.ravel()
```

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

Flatten()

In [ ]:
```
ar.flatten()
```

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

In [ ]:
```
#'C' means to flatten in row-major (C-style) order.
ar.flatten(order= 'C')
```

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

In [ ]:
```
# 'F' means to flatten in column-major (Fortran- style) order.
ar.flatten(order='F')
```

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

In [ ]:
```
# 'A' means to flatten in column-major order if a is Fortran contiguous in memory, row-major order otherwise.
ar.flatten(order='A')
```

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

In [ ]:
```
# 'K' means to flatten a in the order the elements occur in memory. The default is 'C'.
ar.flatten(order='K')
```

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

Transpose()

In [ ]:
```
# convert Row into Column
```

In [ ]:
```
ar
```

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

In [ ]:
```
ar.transpose()
```

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

In [ ]:
```
## short key for transpose
ar.T
```

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

Concatenate & Sort( )

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

In [ ]:
```
a= np.array([1,2,3,4,5,6,7,8,9,10,11,12,13,14])
```

In [ ]:
```
b=np.array([20,30,40,50,60,70,80])
```

In [ ]:
```
c= np.concatenate((a,b))
c
```

Out[ ]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 20, 30, 40,
       50, 60, 70, 80])
```

In [ ]:
```
c.sort()
c
```

Out[ ]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 20, 30, 40,
       50, 60, 70, 80])
```

2-D Arrays

In [ ]:
```
d2 = np.array([[5,67,20],
               [10,20,30],
               [30,40,50]])
d2
```

Out[ ]:
```
array([[ 5, 67, 20],
       [10, 20, 30],
       [30, 40, 50]])
```

In [ ]:
```
d2.ndim
```

Out[ ]:
```
2
```

In [ ]:
```
d3=np.array([[[1,2,3],[1,2,3],[1,2,3]],
             [[1,2,3],[1,2,3],[1,2,3]],[[1,2,3],[1,2,3],[1,2,3]]])
d3
```

Out[ ]:
```
array([[[1, 2, 3],
        [1, 2, 3],
        [1, 2, 3]],

       [[1, 2, 3],
        [1, 2, 3],
        [1, 2, 3]],

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

In [ ]:
```
d3.ndim
```

Out[ ]:
```
3
```

In [ ]:
```
d3.size
```

Out[ ]:
```
27
```

In [ ]:
```
d3.shape
```

Out[ ]:
```
(3, 3, 3)
```

In [ ]:
```
d3.sort()
d3
```

Out[ ]:
```
array([[[1, 2, 3],
        [1, 2, 3],
        [1, 2, 3]],

       [[1, 2, 3],
        [1, 2, 3],
        [1, 2, 3]],

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

In [ ]:
```
new=np.arange(9)
new
```

Out[ ]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
```

In [ ]:
```
new.reshape(3,3)
```

Out[ ]:
```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

In [ ]:
```
np.reshape(new, newshape=(1,9),order="F")
```

Out[ ]:
```
array([[0, 1, 2, 3, 4, 5, 6, 7, 8]])
```

Reshape with New Axis

In [ ]:
```
a=np.arange(1,10)
a
```

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

In [ ]:
```
# Row Wise
b=a[np.newaxis, :]
b
```

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

In [ ]:
```
# Column Wise
c=a[:,np.newaxis]
c
```

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

In [ ]:
```
a[2:5]
```

Out[ ]:
```
array([3, 4, 5])
```

In [ ]:
```

```

Python Numpy Mathematical Operations

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

In [ ]:
```
arr1 = np.arange(1,10).reshape(3,3)
arr1
```

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

In [ ]:
```
arr2 = np.arange(1,10).reshape(3,3)
arr2
```

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

In [ ]:
```
arr1,arr2
```

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

In [ ]:
```
arr1 + arr2
```

Out[ ]:
```
array([[ 2,  4,  6],
       [ 8, 10, 12],
       [14, 16, 18]])
```

In [ ]:
```
np.add(arr1, arr2)
```

Out[ ]:
```
array([[ 2,  4,  6],
       [ 8, 10, 12],
       [14, 16, 18]])
```

In [ ]:
```
arr1 - arr2
```

Out[ ]:
```
array([[0, 0, 0],
       [0, 0, 0],
       [0, 0, 0]])
```

In [ ]:
```
np.subtract(arr1, arr2)
```

Out[ ]:
```
array([[0, 0, 0],
       [0, 0, 0],
       [0, 0, 0]])
```

In [ ]:
```
arr1 / arr2
```

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

In [ ]:
```
np.divide(arr1, arr2)
```

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

In [ ]:
```
arr1 * arr2
```

Out[ ]:
```
array([[ 1,  4,  9],
       [16, 25, 36],
       [49, 64, 81]])
```

In [ ]:
```
arr1, arr2
```

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

In [ ]:
```
np.multiply(arr1, arr2)
```

Out[ ]:
```
array([[ 1,  4,  9],
       [16, 25, 36],
       [49, 64, 81]])
```

In [ ]:
```
arr1, arr2
```

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

In [ ]:
```
## Matrix Product
arr1 @ arr2
```

Out[ ]:
```
array([[ 30,  36,  42],
       [ 66,  81,  96],
       [102, 126, 150]])
```

In [ ]:
```
## mATRIX PRODUCT
np.dot(arr1, arr2)
```

Out[ ]:
```
array([[ 30,  36,  42],
       [ 66,  81,  96],
       [102, 126, 150]])
```

In [ ]:
```
arr1
```

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

Find Maximum & Minimum Value

In [ ]:
```
## find max value in matrics
```

In [ ]:
```
arr1.max()
```

Out[ ]:
```
9
```

In [ ]:
```
## find max value index number
```

In [ ]:
```
arr1.argmax()
```

Out[ ]:
```
8
```

In [ ]:
```
arr1
```

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

Find Mini & Max Value in ROW & COL

In [ ]:
```
## 0 represent Column & 1 Represent Row in Axis value
arr1.max(axis = 0)
```

Out[ ]:
```
array([7, 8, 9])
```

In [ ]:
```
## max value in Row
arr1.max(axis = 1)
```

Out[ ]:
```
array([3, 6, 9])
```

In [ ]:
```
## Minimum Value
```

In [ ]:
```
arr1.min()
```

Out[ ]:
```
1
```

In [ ]:
```
## Minimum value Index
arr1.argmin()
```

Out[ ]:
```
0
```

In [ ]:
```
## find Max value in every Row and Column
```

In [ ]:
```
arr1.min(axis = 0)
```

Out[ ]:
```
array([1, 2, 3])
```

In [ ]:
```
arr1.min(axis = 1)
```

Out[ ]:
```
array([1, 4, 7])
```

In [ ]:
```
arr1
```

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

In [ ]:
```
## Total Sum of Matrics arr1
np.sum(arr1)
```

Out[ ]:
```
45
```

In [ ]:
```
## Sum of each Column
np.sum(arr1, axis = 0)
```

Out[ ]:
```
array([12, 15, 18])
```

In [ ]:
```
## Sum of each Row value
np.sum(arr1, axis=1)
```

Out[ ]:
```
array([ 6, 15, 24])
```

Mean ,Standard Devi, squrt ,Log

In [ ]:
```
arr1
```

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

In [ ]:
```
## Mean value of Matrics
np.mean(arr1)
```

Out[ ]:
```
5.0
```

In [ ]:
```
## squre root of each value in matrics
np.sqrt(arr1)
```

Out[ ]:
```
array([[1.        , 1.41421356, 1.73205081],
       [2.        , 2.23606798, 2.44948974],
       [2.64575131, 2.82842712, 3.        ]])
```

In [ ]:
```
## Standard Deviation
np.std(arr1)
```

Out[ ]:
```
2.581988897471611
```

In [ ]:
```
## Exponent e^x
np. exp(arr1)
```

Out[ ]:
```
array([[2.71828183e+00, 7.38905610e+00, 2.00855369e+01],
       [5.45981500e+01, 1.48413159e+02, 4.03428793e+02],
       [1.09663316e+03, 2.98095799e+03, 8.10308393e+03]])
```

In [ ]:
```
## Log
np.log(arr1)
```

Out[ ]:
```
array([[0.        , 0.69314718, 1.09861229],
       [1.38629436, 1.60943791, 1.79175947],
       [1.94591015, 2.07944154, 2.19722458]])
```

In [ ]:
```
np.log10(arr1)
```

Out[ ]:
```
array([[0.        , 0.30103   , 0.47712125],
       [0.60205999, 0.69897   , 0.77815125],
       [0.84509804, 0.90308999, 0.95424251]])
```

In [ ]:
```

```

SORTING

String Operations Comparison & Information

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

In [ ]:
```
str_1 = ' Learning python Numpy'
str_2 = ' seems like dificult'
```

In [ ]:
```
np.char.add(str_1, str_2)
```

Out[ ]:
```
array(' Learning python Numpy seems like dificult', dtype='<U42')
```

In [ ]:
```
## Lower Letter
np.char.lower(str_1)
```

Out[ ]:
```
array(' learning python numpy', dtype='<U22')
```

In [ ]:
```
##upper case
np.char.upper(str_1)
```

Out[ ]:
```
array(' LEARNING PYTHON NUMPY', dtype='<U22')
```

In [ ]:
```
np.char.center(str_1, 60, fillchar="^")
```

Out[ ]:
```
array('^^^^^^^^^^^^^^^^^^ Learning python Numpy^^^^^^^^^^^^^^^^^^',
      dtype='<U60')
```

In [ ]:
```
np.char.split(str_1)
```

Out[ ]:
```
array(list(['Learning', 'python', 'Numpy']), dtype=object)
```

In [ ]:
```
np.char.splitlines("hello\sami")
```

Out[ ]:
```
array(list(['hello\\sami']), dtype=object)
```

In [ ]:
```
str4= "day"
str5= "date"
np.char.join([":", "/"],[str4, str5])
```

Out[ ]:
```
array(['d:a:y', 'd/a/t/e'], dtype='<U7')
```

In [ ]:
```
np.char.replace(str_1, "Numpy", "Altobalto")
```

Out[ ]:
```
array(' Learning python Altobalto', dtype='<U26')
```

In [ ]:
```
## find string rwual or not
np.char.equal(str4,str5)
```

Out[ ]:
```
array(False)
```

In [ ]:
```
## Find out any char in a string
np.char.count(str_1, "a")
```

Out[ ]:
```
array(1)
```

In [ ]:
```
str_1
```

Out[ ]:
```
' Learning python Numpy'
```

In [ ]:
```
np.char.find(str_1, "Numpy")
```

Out[ ]:
```
array(17)
```

In [ ]:
```

```

Find Trignometry Sin(),Cos() & Tan() Using NumPy

Trignometry Functions & Random Sampling

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [ ]:
```
np.sin(180)
```

Out[ ]:
```
-0.8011526357338304
```

In [ ]:
```
np.cos(180)
```

Out[ ]:
```
-0.5984600690578582
```

In [ ]:
```
np.tan(180)
```

Out[ ]:
```
1.3386902103511544
```

In [ ]:
```
x_sin = np.arange(0, 3*np.pi, 0.1)
```

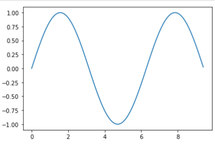
In [ ]:
```
x_sin
```

Out[ ]:
```
array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. , 1.1, 1.2,
       1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2. , 2.1, 2.2, 2.3, 2.4, 2.5,
       2.6, 2.7, 2.8, 2.9, 3. , 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8,
       3.9, 4. , 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 5. , 5.1,
       5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 6. , 6.1, 6.2, 6.3, 6.4,
       6.5, 6.6, 6.7, 6.8, 6.9, 7. , 7.1, 7.2, 7.3, 7.4, 7.5, 7.6, 7.7,
       7.8, 7.9, 8. , 8.1, 8.2, 8.3, 8.4, 8.5, 8.6, 8.7, 8.8, 8.9, 9. ,
       9.1, 9.2, 9.3, 9.4])
```

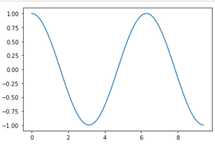
In [ ]:
```
y_sin = np.sin(x_sin)
y_sin
```

Out[ ]:
```
array([ 0.        ,  0.09983342,  0.19866933,  0.29552021,  0.38941834,
        0.47942554,  0.56464247,  0.64421769,  0.71735609,  0.78332691,
        0.84147098,  0.89120736,  0.93203909,  0.96355819,  0.98544973,
        0.99749499,  0.9995736 ,  0.99166481,  0.97384763,  0.94630009,
        0.90929743,  0.86320937,  0.8084964 ,  0.74570521,  0.67546318,
        0.59847214,  0.51550137,  0.42737988,  0.33498815,  0.23924933,
        0.14112001,  0.04158066, -0.05837414, -0.15774569, -0.2555411 ,
       -0.35078323, -0.44252044, -0.52983614, -0.61185789, -0.68776616,
       -0.7568025 , -0.81827711, -0.87157577, -0.91616594, -0.95160207,
       -0.97753012, -0.993691  , -0.99992326, -0.99616461, -0.98245261,
       -0.95892427, -0.92581468, -0.88345466, -0.83226744, -0.77276449,
       -0.70554033, -0.63126664, -0.55068554, -0.46460218, -0.37387666,
       -0.2794155 , -0.1821625 , -0.0830894 ,  0.0168139 ,  0.1165492 ,
        0.21511999,  0.31154136,  0.40484992,  0.49411335,  0.57843976,
        0.6569866 ,  0.72896904,  0.79366786,  0.85043662,  0.8987081 ,
        0.93799998,  0.96791967,  0.98816823,  0.99854335,  0.99894134,
        0.98935825,  0.96988981,  0.94073056,  0.90217183,  0.85459891,
        0.79848711,  0.7343971 ,  0.66296923,  0.58491719,  0.50102086,
        0.41211849,  0.31909836,  0.22288991,  0.12445442,  0.02477543])
```

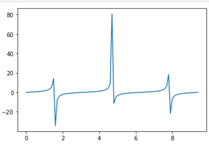
In [ ]:
```
plt.plot(x_sin, y_sin)
plt.show()
```



In [ ]:
```
y_cos = np.cos(x_sin)
plt.plot(x_sin, y_cos)
plt.show()
```



In [ ]:
```
y_tan = np.tan(x_sin)
plt.plot(x_sin, y_tan)
plt.show()
```



Random Sampling With Numpy

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

In [ ]:
```
np.random.random(1)
```

Out[ ]:
```
array([0.8027798])
```

In [ ]:
```
np.random.random((3, 3))
```

Out[ ]:
```
array([[0.78412146, 0.26106899, 0.07138804],
       [0.54579565, 0.92090285, 0.58617832],
       [0.14585943, 0.44304812, 0.5926764 ]])
```

In [ ]:
```
np.random.randint(1, 4)
```

Out[ ]:
```
3
```

In [ ]:
```
## np.random. rand int (values, shape)
np.random.randint(1, 4, (4,4))
```

Out[ ]:
```
array([[1, 2, 2, 1],
       [3, 3, 2, 2],
       [1, 2, 2, 1],
       [3, 2, 2, 3]])
```

In [ ]:
```
np.random.randint(1, 4, (2,4,4))
```

Out[ ]:
```
array([[[2, 1, 1, 1],
        [1, 1, 3, 2],
        [3, 2, 3, 2],
        [1, 3, 2, 3]],

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

In [ ]:
```
np.random.seed(10)
np.random.randint(1, 4, (2,4,4))
```

Out[ ]:
```
array([[[2, 2, 1, 1],
        [2, 1, 2, 2],
        [1, 2, 2, 3],
        [1, 2, 1, 3]],

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

In [ ]:
```
np.random.rand(3)
```

Out[ ]:
```
array([0.13145815, 0.41366737, 0.77872881])
```

In [ ]:
```
np.random.rand(3,3)
```

Out[ ]:
```
array([[0.58390137, 0.18263144, 0.82608225],
       [0.10540183, 0.28357668, 0.06556327],
       [0.05644419, 0.76545582, 0.01178803]])
```

In [ ]:
```
np.random.randn(3,3)
```

Out[ ]:
```
array([[-1.58494101,  1.05535316, -1.92657911],
       [ 0.69858388, -0.74620143, -0.15662666],
       [-0.19363594,  1.13912535,  0.36221796]])
```

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

Out[ ]:
```
1
```

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

Out[ ]:
```
1
```

In [ ]:
```
for i in range(20):
   print(np.random.choice(x))
```

```
1
2
1
4
1
1
3
1
3
3
1
4
2
1
4
2
3
3
2
3
```

In [ ]:
```
x
```

Out[ ]:
```
[1, 2, 3, 4]
```

In [ ]:
```
np.random.permutation(x)
```

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

In [ ]:
```
## Scipy.org for further Radom Function
```