numpy-start-1

August 2, 2024

1 NUMPY 1

Importing Library numpy

Check version of numpy

1D Array

Operations on 1D array

Tasks to solve

[1]:
```
import numpy as np
np.__version__      # numpy = numerical python
```

[1]:
```
'2.0.1'
```

[2]:
```
arr=np.array([25,10,20,30,40,50,60])

print(arr)
```

```
[25 10 20 30 40 50 60]
```

[3]:
```
type(arr) #ndarray = n dimensional array
```

[3]:
```
numpy.ndarray
```

[4]:
```
arr.ndim   #function to check dimension
```

[4]:
```
1
```

[5]:
```
arr.size # size elements of array len(arr)
```

[5]:
```
7
```

[6]:
```
arr.shape   #describe m*n
```

[6]:
```
(7,)
```

[7]:
```
arr.dtype # datatype of array
```

[7]:
```
dtype('int64')
```

[8]:
```
array=np.array([1,2,3,4,5,6.6])  #because of a single float variable all␣
  ↪convert to float
array
```

[8]:
```
array([1. , 2. , 3. , 4. , 5. , 6.6])
```

[9]:
```
array.dtype
```

[9]:
```
dtype('float64')
```

[10]:
```
array1=np.array([1,2,3,4.5,"rahul"])  #because of a single string variable all␣
  ↪convert to string
array1
```

[10]:
```
array(['1', '2', '3', '4.5', 'rahul'], dtype='<U32')
```

[11]:
```
array1.dtype
```

[11]:
```
dtype('<U32')
```

[12]:
```
arr[2]
```

[12]:
```
np.int64(20)
```

[13]:
```
print(arr[-3:])
print(arr[4::])
print(arr[-1:-4:-1]) # 1 shows +ve direction and -1 shows negative direction
print(arr[-3::1])
```

```
[40 50 60]
[40 50 60]
[60 50 40]
[40 50 60]
```

[14]:
```
# addition
arr+5
```

[14]:
```
array([30, 15, 25, 35, 45, 55, 65])
```

[15]:
```
#subtraction
arr-5
```

[15]:
```
array([20,  5, 15, 25, 35, 45, 55])
```

[16]:
```
#divide
arr/5
```

[16]:
```
array([ 5.,  2.,  4.,  6.,  8., 10., 12.])
```

[17]:
```
max(arr)
```

[17]:
```
np.int64(60)
```

[18]:
```
min(arr)
```

[18]:
```
np.int64(10)
```

[19]:
```
np.mean(arr) # mean=avg
```

[19]:
```
np.float64(33.57142857142857)
```

[20]:
```
np.median(arr) # median = center number
```

[20]:
```
np.float64(30.0)
```

[21]:
```
np._Mode(arr)  # numpy has no mode attribute
#mode= frequency of data
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[21], line 1
----> 1 np._Mode(arr)  # numpy has no mode attribute
      2 #mode= frequency of data

File c:\Users\HARSH KUMAR␣
 ↪SAINI\AppData\Local\Programs\Python\Python312\Lib\site-packages\numpy\__init__.
 ↪py:424, in __getattr__(attr)
    421     import numpy.char as char
    422     return char.chararray
--> 424 raise AttributeError("module {!r} has no attribute "
    425                      "{!r}".format(__name__, attr))

AttributeError: module 'numpy' has no attribute '_Mode'
```

[ ]:
```
np.random.randint(2,300,10) # means array of 10 items between 2-300
```

[ ]:
```
array([264, 209, 181, 294, 276,  17, 105,  60, 197,  95], dtype=int32)
```

[ ]:
```
# to generate m*n array
np.random.rand(2,3)
```

```
[ ]: array([[0.18929289, 0.62368485, 0.04484459],
            [0.92248238, 0.47979056, 0.64778753]])
```

```
[ ]: arr.dtype='int32'
     arr.dtype  # that's way we can change datatype
```

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

```
[ ]: import random,string

     string.punctuation
```

```
[ ]: '!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

```
[ ]: string.digits
```

```
[ ]: '0123456789'
```

```
[ ]: string.ascii_letters
```

```
[ ]: 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

```
[ ]: alpha= string.ascii_letters+string.digits
     alpha
```

```
[ ]: 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'
```

```
[ ]: alpha.split
```

```
[ ]: <function str.split(sep=None, maxsplit=-1)>
```

```
[ ]: import numpy as np
     arr1 =np.random.randint(1,4500,100)
     print(arr1)
```

```
[ 953 4141 1144 2131 3512 3288   48 1411 4352 4062 1476 2094  567  490
   39 1746 1051 3192  803 3228 1578 3637 4283 3763  386 1488  222 4497
  726 2486  323  372 4013 2949 1995 2613 4353 3259   73  663 2748 4330
 3791  325 1375 2473  376 2635 1531  339 3659  503  535 4170 2174 3298
 1359 1470 2538 2441 2876  955 3815  265 3941 2715 2050 4459 2883   84
 1290 2204  847 3290   73 1986 2023 2978 3752  309 1124  279 1951 3309
  585 1309 1738 1328 2138 3063 4223 2746 4383 4434 1951 3496 2815 4280
 3157 1365]
```

TASKS 1. Try to calculate no of items <= 100 (try diff methods) 2. Try to calculate no of even no.

1. Try to calculate no of items <= 100 (try diff methods)

[ ]:

```
# 1. try to calculate no of items <= 100 ( try diff methods)

        # method 1 = Using loop
count=0
for i in range(0,100):
    if arr1[i] <= 100:
        print(arr1[i],end=" ")
        count+=1
print("\nNo of elements less then 100 :->  ",count)
```

```
48 39 73 84 73
No of elements less then 100 :->   5
```

[ ]:

```
# Methhod 2= Using sum attribute in array

count=np.sum(arr1<=100)
print("No of elements less then 100 are :-> ",count)
```

```
No of elements less then 100 are :->  5
```

[ ]:

```
# Method 3 = using count_nonzero attribute
count=np.count_nonzero(arr1<=100)
print(count)
```

```
5
```

2. Try to calculate no of even no.

[ ]:

```
# Method 1 = Using loop

count=0
for i in range(0,100):
    if arr1[i]%2==0:
        print(arr1[i],end=" ")
        count+=1
print("\nEven numbers present in random array:->  ",count)
```

```
1144 3512 3288 48 4352 4062 1476 2094 490 1746 3192 3228 1578 386 1488 222 726
2486 372 2748 4330 376 4170 2174 3298 1470 2538 2876 2050 84 1290 2204 3290 1986
2978 3752 1124 1738 1328 2138 2746 4434 3496 4280
Even numbers present in random array:->   44
```

[ ]:

```
# Method 2= using sum attribute
count=np.sum(arr1%2==0)
print("even numbers are :> ",count)
```

```
even numbers are :>  44
```

```
[ ]: #     Method 3 = using count_nonzero attribute
     count=np.count_nonzero(arr1%2==0)
     print(count)
```

```
44
```

numpy-start-2

August 2, 2024

1 Numpy 2

2D Array

3D Array

Accessing Elements in every type of array

Conversion 3D TO 1D

2D ARRAY

[1]:
```
import numpy as np
```

[2]:
```
arr=np.array([[1,2,3],[4,5,6],[7,8,9]])
print(arr)
```

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

[3]:
```
type(arr)
```

[3]:
```
numpy.ndarray
```

[4]:
```
arr.ndim
```

[4]:
```
2
```

[5]:
```
arr.shape
```

[5]:
```
(3, 3)
```

[6]:
```
print(arr[1,2])
print(arr[-2,-2])
```

```
6
5
```

[7]:
```
print(arr*2)
```

```
[[ 2  4  6]
 [ 8 10 12]
 [14 16 18]]
```

[8]:

```
print(arr+2)
```

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

[9]:

```
print(arr[0:2])
```

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

[10]:

```
print(arr [0:2,0:2])
```

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

[11]:

```
print(arr[0:2,1:])
```

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

[12]:

```
print(arr[1:,:2])
```

```
[[4 5]
 [7 8]]
```

[13]:

```
print(arr[-2:,-3:])
```

```
[[4 5 6]
 [7 8 9]]
```

[14]:

```
arr[-2:]
```

[14]:

```
array([[4, 5, 6],
       [7, 8, 9]])
```

[15]:

```
print(arr[:,1:])
```

```
[[2 3]
 [5 6]
 [8 9]]
```

SOME FUNCTIONS

```
[16]: print(np.ones(10)) #to generate quick array
      print(np.zeros(10))
```

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

```
[17]: np.linspace(1,50,10) # 1 to 50 , 10 item ( uniform distribution )
```

```
[17]: array([ 1.        ,  6.44444444, 11.88888889, 17.33333333, 22.77777778,
             28.22222222, 33.66666667, 39.11111111, 44.55555556, 50.        ])
```

3D ARRAY

3D array have multiple two dimensional arrays(Matrix)

```
[18]: arr3=np.
       ↪array([[[1,2,3,4],[1,3,4,5],[1,2,3,4]],[[1,2,3,4],[3,5,6,7],[5,6,7,8]],[[0,9,8,7],[9,8,7,6]
```

```
[19]: arr3.ndim
```

```
[19]: 3
```

```
[20]: arr3
```

```
[20]: array([[[1, 2, 3, 4],
              [1, 3, 4, 5],
              [1, 2, 3, 4]],

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

             [[0, 9, 8, 7],
              [9, 8, 7, 6],
              [7, 6, 5, 4]]])
```

```
[21]: arr3.shape   # 3= table , 3 rows, 4 columns
```

```
[21]: (3, 3, 4)
```

```
[22]: arr3[1]  #access 1st table
```

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

```
[23]: arr3[0:2] # access table 1 and 2
```

```
[23]: array([[[1, 2, 3, 4],
              [1, 3, 4, 5],
              [1, 2, 3, 4]],

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

```
[24]: arr3[0:3:2] #access table 1st and 3rd
```

```
[24]: array([[[1, 2, 3, 4],
              [1, 3, 4, 5],
              [1, 2, 3, 4]],

             [[0, 9, 8, 7],
              [9, 8, 7, 6],
              [7, 6, 5, 4]]])
```

```
[25]: arr3[1,:2] #access 1st table 2 rows
```

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

```
[26]: arr3[0:4,0,:] #access ever table's first row
```

```
[26]: array([[1, 2, 3, 4],
             [1, 2, 3, 4],
             [0, 9, 8, 7]])
```

```
[27]: arr3[:,-3,:]
```

```
[27]: array([[1, 2, 3, 4],
             [1, 2, 3, 4],
             [0, 9, 8, 7]])
```

```
[28]: arr3[2,1:,1:] #access 3rd tables small matrix
```

```
[28]: array([[8, 7, 6],
             [6, 5, 4]])
```

```
[29]: arr3[0,1:,1:] #acess with +ve index
```

```
[29]: array([[3, 4, 5],
             [2, 3, 4]])
```

```
[30]: arr3[-3,-2:,-3:] #acess with -ve index
```

```
[30]: array([[3, 4, 5],
             [2, 3, 4]])
```

```
[31]: arr3[:,1:,1:] #acess ever tables's last 2 rows and 3 columns
```

```
[31]: array([[[3, 4, 5],
              [2, 3, 4]],

             [[5, 6, 7],
              [6, 7, 8]],

             [[8, 7, 6],
              [6, 5, 4]]])
```

```
[32]: arr3[:,-2:,-3:] # with -ve index
```

```
[32]: array([[[3, 4, 5],
              [2, 3, 4]],

             [[5, 6, 7],
              [6, 7, 8]],

             [[8, 7, 6],
              [6, 5, 4]]])
```

```
[33]: arr3[:,-1,::-1]  # access every tables's last row in reverse order
```

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

Convert 3D array in 1D array

By ravel() function .

```
[34]: arr3.ravel()
```

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

By flatten ()

```
[35]: arr3.flatten()
```

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

By resahpe( )

[36]:
```
arr3.reshape(36) #(3,3,4)=39
```

[36]:
```
array([1, 2, 3, 4, 1, 3, 4, 5, 1, 2, 3, 4, 1, 2, 3, 4, 3, 5, 6, 7, 5, 6,
       7, 8, 0, 9, 8, 7, 9, 8, 7, 6, 7, 6, 5, 4])
```

numpy-start-3

August 2, 2024

1 Numpy 3

argmin

argmax

argsort

array generate br range

Generate Diagonal array

generate array elements with fix difference

Various random functions - rand(), randn(), randf(), randint()

additional operations on array - sqroot() , sin() & cos() , cumsum()

Broadcasting in numpy

COPY and VIEW

Difference between copy and view

Join array

Stack attribute

Split array

searching in array

sorting in array

@ Use of argmin

[99]:

```python
import numpy as np
```

[100]:

```python
arr=np.array([10,2,3,4])
index_of_minimum=arr.argmin()
print(index_of_minimum)
```

```
1
```

@ Use of argmax

[101]:

```
arr=np.array([10,2,3,4])
index_of_maximum=arr.argmax()
print(index_of_maximum)
```

```
0
```

@ Use of argsort

[102]:

```
index__sort=arr.argsort()
print(index__sort)
```

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

[103]:

```
# Generate dimensiona as want

arr1=np.array([1,2,3,4],ndmin=10)
print(arr1)
print(arr1.ndim)
```

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

generate array as want

[104]:

```
arr2=np.arange(3,9)  # by this the array will be created from 3 to 8
arr2
```

[104]:

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

Generate Diagonal array

[105]:

```
arr3=np.eye(3)  # 3*3 dimensional array is created using 'eye'
arr3
```

[105]:

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

[106]:

```
arr4=np.eye(4,7)
arr4
```

[106]:

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

generate array elements with fix difference

[107]:
```
arr_lin=np.linspace(0,20,num=8)
arr_lin
```

[107]:
```
array([ 0.        ,  2.85714286,  5.71428571,  8.57142857, 11.42857143,
       14.28571429, 17.14285714, 20.        ])
```

1.1 Various random functions

1. rand() = This function is used to generate random vlaues between 0 and 1

[108]:
```
var=np.random.rand()    # generate random value
print(var)

var2=np.random.rand(2,5)   # generate a array  of 2*3 with random values between␣
 ↪0-1
print(var2)
```

```
0.2226592783706064
[[0.64234688 0.60747293 0.32413822 0.72653636 0.2715    ]
 [0.2027201  0.41506867 0.89861721 0.86063204 0.53695481]]
```

2. randn() = This function is used to generate random vlaues closest to 0. May return +ve or -ve as well

[109]:
```
var3=np.random.randn()   # generate random value
print(var3)

var4=np.random.randn(4,6)    #generate array
print(var4)
```

```
1.2290869247403715
[[-0.28743706 -1.56082372 -0.95363285  0.78298109 -0.2378703   0.3774634 ]
 [-2.20908408  0.23544794  0.67589899  0.87965212 -2.30591087  1.59444663]
 [ 0.50585666  0.67135258  0.65810602  1.3928001   0.314703   -0.23325492]
 [ 1.06001312  0.73399892 -0.06844504  0.59889168 -0.34502826  1.11833283]]
```

3. ranf() = it returns 1D array with specific shape filled with floats between range [0.0,1.0)

[110]:
```
var5=np.random.ranf(4)
var5
```

[110]:
```
array([0.04261696, 0.33812899, 0.11396558, 0.38625973])
```

4. Randint() = It generate a array filled with elements between given range

[111]:
```
var6=np.random.randint(1,90)   #  returns a single element between range 1-90
print(var6)
```

```
var7=np.random.randint(1,50,20) # returns a array of 20 element between range␣
 ↪1-90
print(var7)
```

```
79
[32 49 25 48 43 46 19 43 43  3 21 41 23 49 14 13 12 29 43  2]
```

1.2 Some additional operations on array

1. sqroot
2. sin and cos
3. cumsum

1. sqroot

```
[112]: arr1=[1,2,3,4,5,6]
       sqr=np.sqrt(arr1)    # sqrt used to retuen square root of every element
       sqr
```

```
[112]: array([1.        , 1.41421356, 1.73205081, 2.        , 2.23606798,
              2.44948974])
```

2. sin and cos

```
[113]: print(np.sin(arr1))   # done sin function on every element of arr1 array
       print(np.cos(arr1))   # done cos function on every element of arr1 array
```

```
[ 0.84147098  0.90929743  0.14112001 -0.7568025  -0.95892427 -0.2794155 ]
[ 0.54030231 -0.41614684 -0.9899925  -0.65364362  0.28366219  0.96017029]
```

3. cumsum

```
[114]: print(np.cumsum(arr1))  # returns the cumulative sum of the elements . like if␣
        ↪arr1=[1,2,3,4,5,6] then 1,1+2=3,3+3=6,6+4=10,10+5=15,15+6=21
```

```
[ 1  3  6 10 15 21]
```

1.3 Broadcasting in numpy

For 1D array

- addition of two 1D arrays

```
[115]: arr1=np.array([1,2,3,4])
       arr2=np.array([4,5,6])

       print(arr1+arr2)          # so to add two 1D arrays , dimension(shape) of both␣
        ↪must be same otherwise it will give ValueError
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
```

```
Cell In[115], line 4
      1 arr1=np.array([1,2,3,4])
      2 arr2=np.array([4,5,6])
----> 4 print(arr1+arr2)           # so to add two 1D arrays , dimension(shape)␣
 ↪of both must be same otherwise it will give ValueError

ValueError: operands could not be broadcast together with shapes (4,) (3,)
```

For 2D array

- for two 2d array in first shape is (m * __) and second is (__ * m)

[ ]:

```python
var1=np.array([1,2,3,4,8])          # var1 shape is 1*_  ( 1 row and 4 column )
var2=np.array([[1],[5],[7],[9],[8]])  # var2 sahpe is _*1 ( 4 row and 1 column )
print(var1)
print()
print(var2)
print()
print(var1+var2)      # (1*4) and (4*1) result will be 4*4
print()
print(var1*var2)
```

```
[1 2 3 4 8]

[[1]
 [5]
 [7]
 [9]
 [8]]

[[ 2  3  4  5  9]
 [ 6  7  8  9 13]
 [ 8  9 10 11 15]
 [10 11 12 13 17]
 [ 9 10 11 12 16]]

[[ 1  2  3  4  8]
 [ 5 10 15 20 40]
 [ 7 14 21 28 56]
 [ 9 18 27 36 72]
 [ 8 16 24 32 64]]
```

2 COPY AND VIEW

1. Copy Copy function is used to copy the existing array

```
[ ]: e_arr=np.array([1,2,3,4])
     copy_arr= np.copy(e_arr)

     print("Real: ",e_arr)
     print("Copy: ",e_arr)
```

```
Real:  [1 2 3 4]
Copy:  [1 2 3 4]
```

2. View view shows the array allocated to it

```
[ ]: e_arr=np.array([10,20,34,48])
     view_arr= e_arr.view()

     print("Real: ",e_arr)
     print("View: ",e_arr)
```

```
Real:  [10 20 34 48]
View:  [10 20 34 48]
```

2.0.1 DIfference between Copy and View

copy- copy owned data from real.

real- real dosen't have it's own data.

copy-copy of a array is new array.

view- a view of original data.

copy- changes in copy array dosen't reflect in original array.

view- changes in view array will reflect in original array.

3 * Join Array

- In this cell we will read about joining of arrays

  1d array

```
[ ]: arr1=np.array([1,2,3,4])
     arr2=np.array([5,6,7])

     j_arr=np.concatenate((arr1,arr2))
     print(j_arr)
```

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

  2D array

```
[ ]: arr1=np.array([[1,2,3,4],[2,7,9,9]])
     arr2=np.array([[5,6,7,5],[2,5,6,0]])

     new_arr=np.concatenate((arr1,arr2),axis=1)  # axis =1 -> arrays wiil be join␣
      ↪accoring to column
     print(arr1)
     print()
     print(arr2)
     print()
     print(new_arr)
```

```
[[1 2 3 4]
 [2 7 9 9]]

[[5 6 7 5]
 [2 5 6 0]]

[[1 2 3 4 5 6 7 5]
 [2 7 9 9 2 5 6 0]]
```

```
[ ]: arr1=np.array([[1,2,3,4],[1,3,5]])  #  To avoid ValueError , setting an array␣
      ↪element with a sequence.
     arr2=np.array([[5,6,7],[1,2,4,5]])

     j_arr=np.concatenate((arr1,arr2))
     print(j_arr)         # this join won't work
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[75], line 1
----> 1 arr1=np.array([[1,2,3,4],[1,3,5]])
      2 arr2=np.array([[5,6,7],[1,2,4,5]])
      4 j_arr=np.concatenate((arr1,arr2))

ValueError: setting an array element with a sequence. The requested array has a␣
 ↪inhomogeneous shape after 1 dimensions. The detected shape was (2,) +␣
 ↪inhomogeneous part.
```

3.1 Stack

- stack besed on join opeartion but retuen pairs in the a array

```
[ ]: arr_1=np.array([1,2,3,4])
     arr_2=np.array([5,6,7,8])

     new_a=np.stack((arr_1,arr_2),axis=1)
     new_a_0=np.stack((arr_1,arr_2),axis=0)   # by default axis always is 0
```

```
new_a_d=np.stack((arr_1,arr_2))
print(new_a)
print()
print(new_a_0)
print()
print(new_a_d)
```

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

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

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

3.2 Split array

- arrays can be split in equal parts by split function

[ ]:

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

n_new=np.split(n_arr,3)   # split the array in 3 parts
# n_new1=np.split(n_arr,4)   # split the array in 4 parts
# n_new2=np.split(n_arr,5)   # split the array in 5 parts
n_new3=np.split(n_arr,2)   # split the array in 2 parts

print(n_new)
print()
# print(n_new1)
# print()
# print(n_new2)
print()
print(n_new3)

# split in 4 and 5 parts can not be possiple. spliting should equal(each part␣
 ↪contain equal elements)
```

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


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

3.3 Searching in numpy

```
[ ]: a1=np.array([1,2,4,6,2,8,9])
     x=np.where(a1==2)        # we can search an element in an array and can get its␣
      ↪index using index
     print(x)
```

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

```
[ ]: a1=np.array([1,2,4,6,2,8,9])
     x=np.where(a1%2==0)
     print(x)
```

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

3.3.1 Search sorted Array

it retrun the index where the specific value would be inserted in sorted array

```
[ ]: a2=np.array([1,2,4,6,7,9])
     x1=np.searchsorted(a2,5) # it shows in a1 what will be the location of 5 (␣
      ↪sorted array)
     x2=np.searchsorted(a2,3) # it shows in a1 what will be the location of 3 (␣
      ↪sorted array)

     print(x1)
     print(x2)
```

```
3
2
```

3.4 SORT

```
[ ]: a1=np.array([2,4,7,1,45,65,1,24,7])
     print(np.sort(a1))
```

```
[ 1  1  2  4  7  7 24 45 65]
```

```
[ ]: a2=np.array(['a','f','g','r'])
     print(np.sort(a2))
```

```
['a' 'f' 'g' 'r']
```