NumPy Sorting

To sort the elements of NumPy array in an ordered sequence numpy.sort() function is used. The parameter arr is mandatory.

If you execute this function on a one-dimensional array, it will return a one-dimensional sorted array containing elements in ascending order.

sort()

> Syntax:
>
> **numpy.sort(arr, axis= -1, kind=None)**

This function allows following Parameters:

- **arr:** Array to be sorted.
- **axis:** This parameter defines the axis along which sorting is performed.

> If this parameter is None, the array will be flattened before sorting, and by default, this parameter is set to -1, which sorts the array along with the last axis.

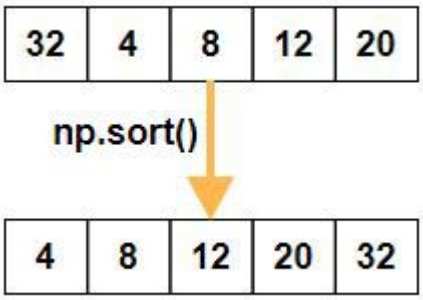
- **kind :** Sorting algorithm ['quicksort'{default}, 'mergesort', 'stable', 'heapsort']

Sorting 1-D Numpy array (Ascending Order)

In 1-D array, it is optional to pass axis parameter to sort() function.

```
import numpy as np

a = np.array([32, 4, 8, 12, 20])
print('Orginal array is', a)
print('Sorted array is', np.sort(a))
```

```
Orginal array is [32  4  8 12 20]
Sorted array is [ 4  8 12 20 32]
```



```
# Import NumPy module
import numpy as np

# Create NumPy array
array = np.array([5,8,6,12,3,15,1])

# To get a sorted array(ascending order)
sorted_array = np.sort(array)
print(sorted_array)
```

```
[ 1  3  5  6  8 12 15]
```

```
import numpy as np

a = np.array(['b', 'a', 'h', 'e', 'k'])

sorted_array = (np.sort(a))
print(sorted_array)
```

```
['a' 'b' 'e' 'h' 'k']
```

```
import numpy as np

a = np.array(['John', 'Micheal', 'George'])

sorted_array = (np.sort(a))
print(sorted_array)
```

```
['George' 'John' 'Micheal']
```

Get A Sorted NumPy Array (Descending Order)

```
# Create NumPy array
array = np.array([5,8,6,12])

# Use numpy.ndarray.sort() to sort
# An array in descending order
array[::-1].sort()
print(array)
```

```
[12  8  6  5]
```

Sorting 2-D Numpy array

What is NumPy axis?

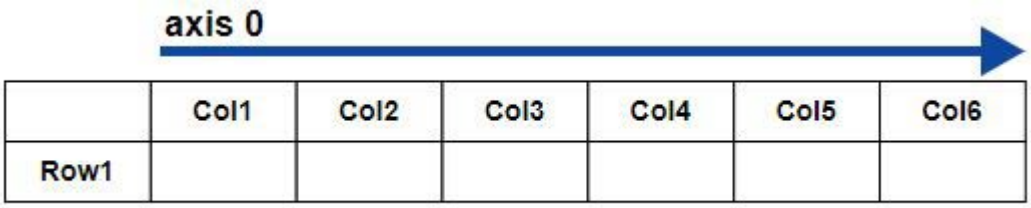
Axes tells the direction along rows and columns.

The number of axes is called dimension of Numpy array.

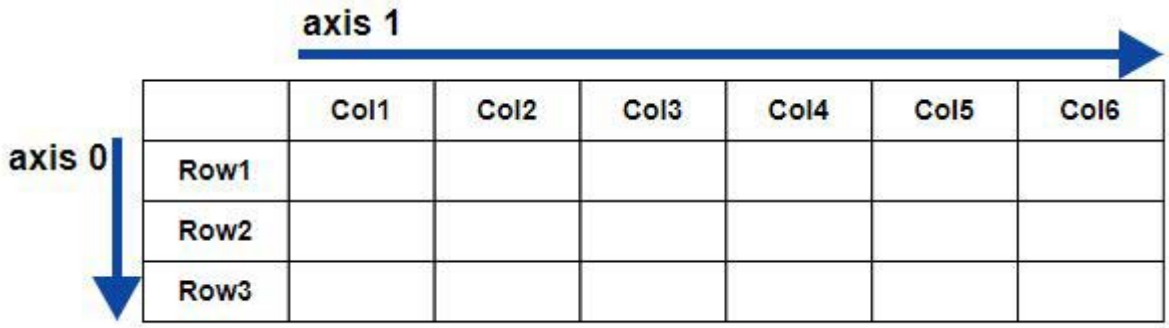
The numbering of the axis starts from 0.

- For 1-D Numpy array, there is only one axis which is axis0.
- For 2-D Numpy array, there are two axes which are- axis0 and axis1.
- For 3-D Numpy array, there are three axes which are- axis0, axis1, and axis2.

In 1-Dimensional array, axis0 goes horizontally across the columns.

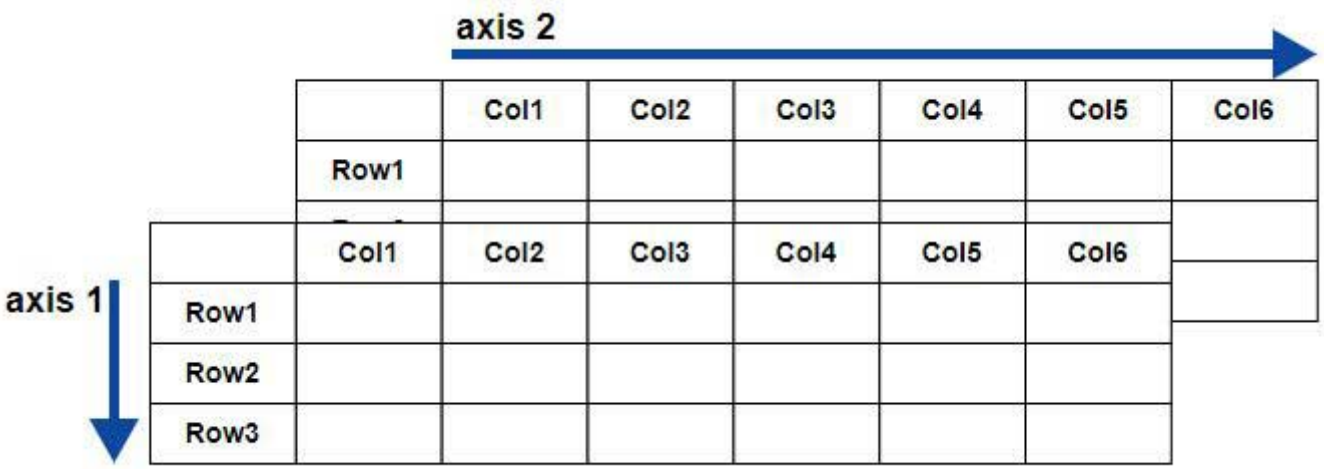


In 2-Dimensional array, axis0 goes vertically across the rows and axis1 goes horizontally across the columns.



3-Dimensional array is simply a collection of 2-Dimensional array.

- **Axis0 goes from one element of 2-Dimensional array to another element present just opposite to another 2-Dimensional array and so on**
- Axis1 goes vertically across the rows and
- Axis2 goes horizontally across the columns.

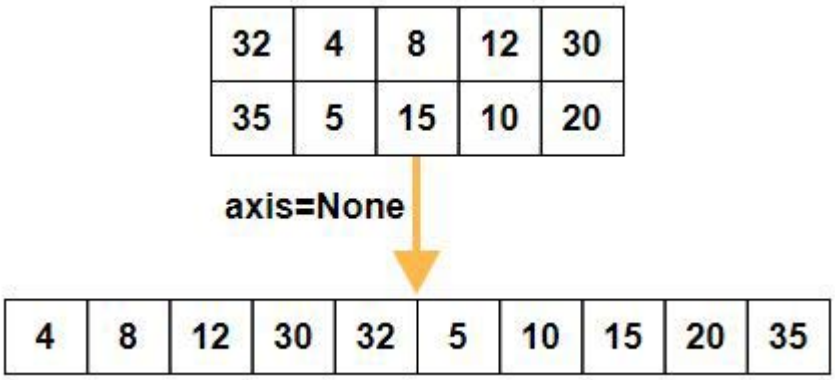


Sorting 2-D Numpy array

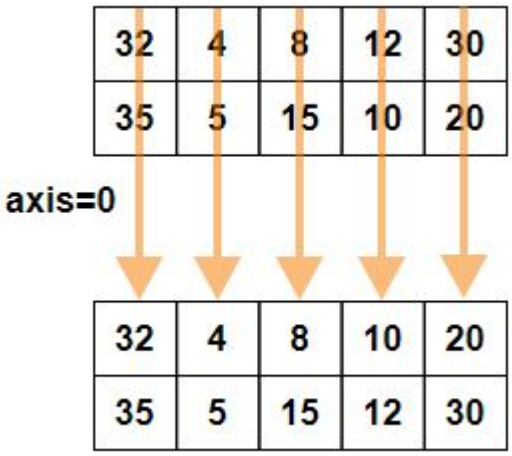
```
import numpy as np

a = np.array([[32, 4, 8, 12, 30], [35, 5, 15, 10, 20]])
print('Sorted array when axis is None\n', np.sort(a, axis=None))
print("\n")
print('SORTING of arr along axis=0 is\n', np.sort(a, axis=0))
print("\n")
print('SORTING of arr along axis=1 is\n', np.sort(a, axis=1))
```

```
Sorted array when axis is None
 [ 4  5  8 10 12 15 20 30 32 35]


Sum of arr along axis=0 is
 [[32  4  8 10 20]
 [35  5 15 12 30]]


Sum of arr along axis=1 is
 [[ 4  8 12 30 32]
 [ 5 10 15 20 35]]
```

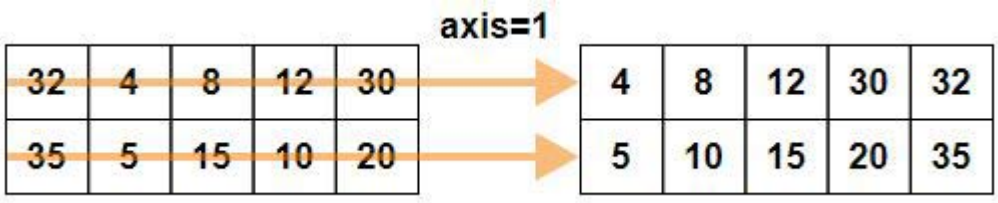
When axis value is None, then sort() function works like this-



When axis=0 is passed to sort() function, then it works like this-



When axis=1 is passed to sort() function, then it works like this-



Sorting 3D NumPy Array

```
import numpy as np

arr = np.array([[[12, 15, 7], [13, 5,11]],[[8, 6, 10],[45,54,70]]])
print("array:\n",arr)


# Use numpy.sort() to Sort a multi-dimensional array
arr2 = np.sort(arr)
print("Sorted array:\n",arr2)


# Sort along the last axis
arr2 = np.sort(arr, axis = -1)
print("Sorted array along axis=-1:\n",arr2)
```

```
array:
 [[[12 15  7]
  [13  5 11]]

 [[ 8  6 10]
  [45 54 70]]]
Sorted array:
 [[[ 7 12 15]
  [ 5 11 13]]

 [[ 6  8 10]
  [45 54 70]]]
Sorted array along axis=-1:
 [[[ 7 12 15]
```

```
 [ 5 11 13]]

[[ 6  8 10]
 [45 54 70]]]
```

Additional Points:

Multi-Dimensional NumPy Arrays Sorting Along Specified Axis

AXIS NONE

```
 # Create NumPy arrays
arr = np.array([[12, 15, 7], [13, 5,11], [8, 6, 10],[45,54,70]])

# Sort multi-dimensional array along a specified axis
arr2 = np.sort(arr, axis= None)
print(" Sorted array:\n",arr2)
```

```
Sorted array:
[ 5  6  7  8 10 11 12 13 15 45 54 70]
```

AXIS 0

```
# Use numpy.sort() to first axis
arr2 = np.sort(arr, axis= 0)
print("Sorted array:\n",arr2)
```

```
Sorted array:
 [[ 8  5  7]
 [12  6 10]
 [13 15 11]
 [45 54 70]]
```

```
import numpy as np

a = np.array([[1,7,4],
              [4,2,2],
              [3,2,8]])
c = np.sort(a, axis=0)
print(c)
```

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

```
import numpy as np
```

```
a = np.array([[1,7,4],
              [4,2,2]])
c = np.sort(a, axis=0)
print(c)
```

```
     [[1 2 2]
      [4 7 4]]
```

AXIS 1

```
 # Create NumPy arrays
arr = np.array([[12, 15, 7], [13, 5,11], [8, 6, 10],[45,54,70]])

# Sort multi-dimensional array along a specified axis
arr2 = np.sort(arr, axis= 1)
print(" Sorted array:\n",arr2)
```

```
      Sorted array:
      [[ 7 12 15]
      [ 5 11 13]
      [ 6  8 10]
      [45 54 70]]
```

```
import numpy as np

a = np.array([[1,7,4],
              [4,2,2],
              [3,2,8]])
c = np.sort(a, axis=1)
print(c)
```

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

Applyting Sorting Algorithms

There are various sorting algorithms like quicksort, merge sort and heapsort which is implemented using the numpy.sort() function.

```
import numpy as np
arr = np.array([2, 1, 4, 3, 5])

#bubble insertion selection merge quick heap
#b = np.sort(arr, kind='insertionsort')
#print(arr)

c = np.sort(arr, kind='selectionsort')
print(c)
print("\n")

e = np.sort(arr, kind='quicksort')
```

```
print(e)
print("\n")

f = np.sort(arr, kind='heapsort')
print(f)
print("\n")

g = np.sort(arr, kind='mergesort')
print(g)
print("\n")
```

```
[1 2 3 4 5]


[1 2 3 4 5]


[1 2 3 4 5]


[1 2 3 4 5]
```

Program to illustrate sorting along different axes using numpy.sort()

```
import numpy as np
#creating an array
A = np.array([[15, 1], [19, 94]])
print ("The input array is : \n", A)
# sorting along the first axis
A_sorted = np.sort(A, axis = 0)
print ("Sorted array along the first axis : \n", A_sorted)
#sorting along the last axis
A_sorted = np.sort(A, axis = -1)
print ("Sorted array along the last axis : \n", A_sorted)
#sorting the flattened axis
A_sorted = np.sort(A, axis = None)
print ("Sorted array when flattened: \n", A_sorted)
```

```
The input array is :
 [[15  1]
 [19 94]]
Sorted array along the first axis :
 [[15  1]
 [19 94]]
Sorted array along the last axis :
 [[ 1 15]
 [19 94]]
Sorted array when flattened:
 [ 1 15 19 94]
```

Program to illustrate sorting using different sorting algorithms using numpy.sort()

Note: 'stable' automatically chooses the best stable sorting algorithm for the data type being sorted.

```
import numpy as np
#creating an array
A = np.array([[19, 3], [19, 94]])
print ("The input array is : \n", A)

# sorting along the first axis using quicksort
A_sorted = np.sort(A, axis = 0, kind = 'quicksort')
print ("Sorted array using quicksort : \n", A_sorted)

# sorting along the first axis using mergesort
A_sorted = np.sort(A, axis = 0, kind = 'mergesort')
print ("Sorted array using mergesort : \n", A_sorted)

# sorting along the first axis using heapsort
A_sorted = np.sort(A, axis = 0, kind = 'heapsort')
print ("Sorted array using heapsort : \n", A_sorted)

# sorting along the first axis using stable
A_sorted = np.sort(A, axis = 0, kind = 'stable')
print ("Sorted array using stable : \n", A_sorted)
```

```
The input array is :
 [[19  3]
 [19 94]]
Sorted array using quicksort :
 [[19  3]
 [19 94]]
Sorted array using mergesort :
 [[19  3]
 [19 94]]
Sorted array using heapsort :
 [[19  3]
 [19 94]]
Sorted array using stable :
 [[19  3]
 [19 94]]
```

✓ 0s completed at 15:47