NumPy - Sort, Search & Counting Functions

A variety of sorting related functions are available in NumPy. These sorting functions implement different sorting algorithms, each of them characterized by the speed of execution, worst case performance, the workspace required and the stability of algorithms. Following table shows the comparison of three sorting algorithms.

kind	speed	worst case	work space	stable
‘quicksort’	1	O(n^2)	0	no
‘mergesort’	2	O(n*log(n))	~n/2	yes
‘heapsort’	3	O(n*log(n))	0	no

numpy.sort()

The sort() function returns a sorted copy of the input array. It has the following parameters −

```
numpy.sort(a, axis, kind, order)
```

Where,

Sr.No.	Parameter & Description
1	**a** Array to be sorted
2	**axis**

	The axis along which the array is to be sorted. If none, the array is flattened, sorting on the last axis
3	**kind** Default is quicksort
4	**order** If the array contains fields, the order of fields to be sorted

Example

Live Demo

```
import numpy as np
a = np.array([[3,7],[9,1]])

print 'Our array is:'
print a
print '\n'

print 'Applying sort() function:'
print np.sort(a)
print '\n'

print 'Sort along axis 0:'
print np.sort(a, axis = 0)
print '\n'

# Order parameter in sort function
dt = np.dtype([('name', 'S10'),('age', int)])
```

```
a = np.array([("raju",21),("anil",25),("ravi", 17), ("amar",27)], dtype = dt)


print 'Our array is:'

print a

print '\n'


print 'Order by name:'

print np.sort(a, order = 'name')
```

It will produce the following output −

```
Our array is:
[[3 7]
 [9 1]]

Applying sort() function:
[[3 7]
 [1 9]]

Sort along axis 0:
[[3 1]
 [9 7]]

Our array is:
[('raju', 21) ('anil', 25) ('ravi', 17) ('amar', 27)]

Order by name:
[('amar', 27) ('anil', 25) ('raju', 21) ('ravi', 17)]
```

numpy.argsort()

The **numpy.argsort()** function performs an indirect sort on input array, along the given axis and using a specified kind of sort to return the array of indices of data. This indices array is used to construct the sorted array.

Example

Live Demo

```
import numpy as np

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


print 'Our array is:'
print x

print '\n'


print 'Applying argsort() to x:'

y = np.argsort(x)

print y

print '\n'


print 'Reconstruct original array in sorted order:'

print x[y]

print '\n'


print 'Reconstruct the original array using loop:'

for i in y:

   print x[i],
```

It will produce the following output −

```
Our array is:
[3 1 2]

Applying argsort() to x:
[1 2 0]

Reconstruct original array in sorted order:
[1 2 3]

Reconstruct the original array using loop:
1 2 3
```

numpy.lexsort()

function performs an indirect sort using a sequence of keys. The keys can be seen as a column in a spreadsheet. The function returns an array of indices, using which the sorted data can be obtained. Note, that the last key happens to be the primary key of sort.

Example

Live Demo

```
import numpy as np


nm = ('raju','anil','ravi','amar')

dv = ('f.y.', 's.y.', 's.y.', 'f.y.')

ind = np.lexsort((dv,nm))


print 'Applying lexsort() function:'

print ind

print '\n'


print 'Use this index to get sorted data:'

print [nm[i] + ", " + dv[i] for i in ind]
```

It will produce the following output −

```
Applying lexsort() function:
[3 1 0 2]

Use this index to get sorted data:
['amar, f.y.', 'anil, s.y.', 'raju, f.y.', 'ravi, s.y.']
```

NumPy module has a number of functions for searching inside an array. Functions for finding the maximum, the minimum as well as the elements satisfying a given condition are available.

numpy.argmax() and numpy.argmin()

These two functions return the indices of maximum and minimum elements respectively along the given axis.

Example

Live Demo

```
import numpy as np

a = np.array([[30,40,70],[80,20,10],[50,90,60]])
```

```python
print 'Our array is:'

print a

print '\n'


print 'Applying argmax() function:'

print np.argmax(a)

print '\n'


print 'Index of maximum number in flattened array'

print a.flatten()

print '\n'


print 'Array containing indices of maximum along axis 0:'

maxindex = np.argmax(a, axis = 0)

print maxindex

print '\n'


print 'Array containing indices of maximum along axis 1:'

maxindex = np.argmax(a, axis = 1)

print maxindex

print '\n'


print 'Applying argmin() function:'

minindex = np.argmin(a)

print minindex

print '\n'
print 'Flattened array:'

print a.flatten()[minindex]

print '\n'



print 'Flattened array along axis 0:'

minindex = np.argmin(a, axis = 0)

print minindex

print '\n'



print 'Flattened array along axis 1:'

minindex = np.argmin(a, axis = 1)

print minindex
```

It will produce the following output −

```
Our array is:
[[30 40 70]
 [80 20 10]
 [50 90 60]]

Applying argmax() function:
7

Index of maximum number in flattened array
[30 40 70 80 20 10 50 90 60]

Array containing indices of maximum along axis 0:
[1 2 0]

Array containing indices of maximum along axis 1:
[2 0 1]

Applying argmin() function:
5

Flattened array:
10

Flattened array along axis 0:
[0 1 1]

Flattened array along axis 1:
[0 2 0]
```

numpy.nonzero()

The **numpy.nonzero()** function returns the indices of non-zero elements in the input array.

Example

Live Demo

```
import numpy as np

a = np.array([[30,40,0],[0,20,10],[50,0,60]])



print 'Our array is:'

print a

print '\n'



print 'Applying nonzero() function:'

print np.nonzero (a)
```

It will produce the following output −

```
Our array is:
[[30 40 0]
 [ 0 20 10]
 [50 0 60]]

Applying nonzero() function:
(array([0, 0, 1, 1, 2, 2]), array([0, 1, 1, 2, 0, 2]))
```

numpy.where()

The where() function returns the indices of elements in an input array where the given condition is satisfied.

Example

Live Demo

```
import numpy as np

x = np.arange(9.).reshape(3, 3)



print 'Our array is:'
print x


print 'Indices of elements > 3'

y = np.where(x > 3)

print y


print 'Use these indices to get elements satisfying the condition'

print x[y]
```

It will produce the following output −

```
Our array is:
[[ 0. 1. 2.]
 [ 3. 4. 5.]
 [ 6. 7. 8.]]

Indices of elements > 3
(array([1, 1, 2, 2, 2]), array([1, 2, 0, 1, 2]))

Use these indices to get elements satisfying the condition
[ 4. 5. 6. 7. 8.]
```

numpy.extract()

The **extract()** function returns the elements satisfying any condition.

Live Demo

```
import numpy as np

x = np.arange(9.).reshape(3, 3)


print 'Our array is:'

print x


# define a condition

condition = np.mod(x,2) == 0


print 'Element-wise value of condition'
print condition



print 'Extract elements using condition'

print np.extract(condition, x)
```

It will produce the following output −

```
Our array is:
[[ 0. 1. 2.]
 [ 3. 4. 5.]
 [ 6. 7. 8.]]

Element-wise value of condition
[[ True False True]
 [False True False]
 [ True False True]]

Extract elements using condition
[ 0. 2. 4. 6. 8.]
```