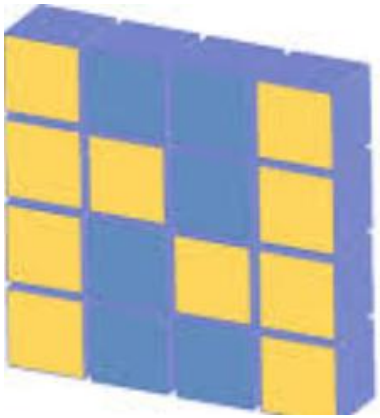
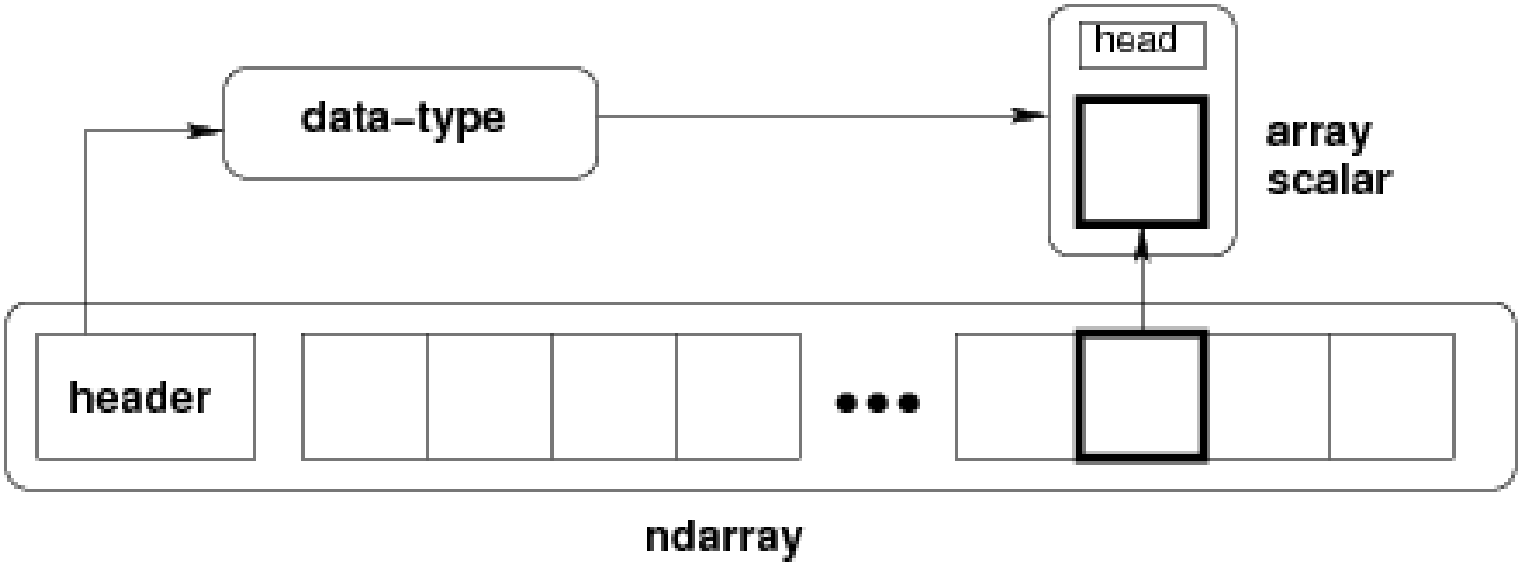
NumPy



NumPy


head
data-type
array
scalar
header
ndarray

```
import numpy as np
a=np.array([1,2,3])
print a
```

The output is as follows:

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

```
# more than one dimensions
import numpy as np
a = np.array([[1, 2], [3, 4]])
print a
```

The output is as follows:

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

```
# minimum dimensions
import numpy as np
a=np.array([1, 2, 3,4,5], ndmin=2)
print a
```

The output is as follows:

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

```
a = np.array([1, 2, 3], dtype=float)
a
```

```
array([ 1.,  2.,  3.])
```

Data Types

Data Types	Description
bool_	Boolean (True or False) stored as a byte
int_	Default integer type (same as C long; normally either int64 or int32)
intc	Identical to C int (normally int32 or int64)
intp	Integer used for indexing (same as C ssize_t; normally either int32 or int64)
int8	Byte (-128 to 127)
int16	Integer (-32768 to 32767)

```
# file name can be used to access content of age column
import numpy as np
dt = np.dtype([('age',np.int8)])
a = np.array([(10,),(20,),(30,)], dtype=dt)
print a['age']
```

The output is as follows:

```
[10 20 30]
```

```
import numpy as np
student=np.dtype([('name','S20'), ('age', 'i1'), ('marks', 'f4')])
a = np.array([('abc', 21, 50),('xyz', 18, 75)], dtype=student)
print a
```

The output is as follows:

```
[('abc', 21, 50.0), ('xyz', 18, 75.0)]
```

ndarray.shape

- This array attribute returns a tuple consisting of array dimensions. It can also be used to resize the array.

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

The output is as follows:

```
(2, 3)
```

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]])
b = a.reshape(3,2)
print b
```

```
# this resizes the ndarray
import numpy as np
a=np.array([[1,2,3],[4,5,6]])
a.shape=(3,2)
print a
```

The output is as follows:

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

ndarray.ndim

- This array attribute returns the number of array dimensions

numpy.itemsize

```
a = np.array([1, 2, 3], dtype=float)
a
```

```
array([ 1.,  2.,  3.])
```

```
a.ndim
```

```
1
```

```
a.itemsize
```

```
8
```

numpy.zeros

- Returns a new array of specified size, filled with zeros

numpy.zeros(shape, dtype=float, order='C')

Shape	Shape of an empty array in int or sequence of int
Dtype	Desired output data type. Optional
Order	'C' for C-style row-major array, 'F' for FORTRAN style column-major array

```
# array of five zeros. Default dtype is float
import numpy as np
x = np.zeros(5)
print x
```

The output is as follows:

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

```
import numpy as np
x = np.zeros((5,), dtype=np.int)
print x
```

Now, the output would be as follows:

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

numpy.ones

- Returns a new array of specified size and type, filled with ones.

Shape	Shape of an empty array in int or tuple of int
Dtype	Desired output data type. Optional
Order	'C' for C-style row-major array, 'F' for FORTRAN style column-major array

```
# array of five ones. Default dtype is float
import numpy as np
x = np.ones(5)
print x
```

The output is as follows:

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

```
import numpy as np
x = np.ones([2,2], dtype=int)
print x
```

Now, the output would be as follows:

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

numpy.asarray

- This function is similar to numpy.array except for the fact that it has fewer parameters. This routine is useful for converting Python sequence into ndarray

numpy.asarray(a, dtype=None, order=None)

a	Input data in any form such as list, list of tuples, tuples, tuple of tuples or tuple of lists
dtype	By default, the data type of input data is applied to the resultant ndarray
order	C (row major) or F (column major). C is default

```
# convert list to ndarray
import numpy as np
x = [1,2,3]
a = np.asarray(x)
print a
```

Its output would be as follows:

```
[1  2  3]
```

```
# ndarray from list of tuples
import numpy as np
x = [(1,2,3),(4,5)]
a = np.asarray(x)
print a
```

Here, the output would be as follows:

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

```
# ndarray from tuple
import numpy as np
x = (1,2,3)
a = np.asarray(x)
print a
```

Its output would be:

```
[1  2  3]
```

numpy.frombuffer

- This function interprets a buffer as one-dimensional array. Any object that exposes the buffer interface is used as parameter to return an **ndarray**

numpy.frombuffer(buffer, dtype=float, count=-1, offset=0)

buffer	Any object that exposes buffer interface
dtype	Data type of returned ndarray. Defaults to float
count	The number of items to read, default -1 means all data
offset	The starting position to read from. Default is 0

```
import numpy as np
s = 'Hello World'
a = np.frombuffer(s, dtype='S1')
print a
```

Here is its output:

```
['H'  'e'  'l'  'l'  'o'  ' '  'W'  'o'  'r'  'l'  'd']
```

numpy.arange

- This function returns an **ndarray object containing evenly spaced values within a given** range.

numpy.arange(start, stop, step, dtype)

start	The start of an interval. If omitted, defaults to 0
stop	The end of an interval (not including this number)
step	Spacing between values, default is 1
dtype	Data type of resulting ndarray. If not given, data type of input is used

```
import numpy as np
x = np.arange(5)
print x
```

Its output would be as follows:

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

```
import numpy as np
# dtype set
x = np.arange(5, dtype=float)
```

```
print x
```

Here, the output would be:

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

```
# start and stop parameters set
import numpy as np
x = np.arange(10,20,2)
print x
```

Its output is as follows:

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

numpy.linspace

- This function is similar to **arange() function. In this function, instead of step size, the** number of evenly spaced values between the interval is specified.

numpy.linspace(start, stop, num, endpoint, retstep, dtype)

start	The starting value of the sequence
stop	The end value of the sequence, included in the sequence if endpoint set to true
num	The number of evenly spaced samples to be generated. Default is 50
endpoint	True by default, hence the stop value is included in the sequence. If false, it is not included
retstep	If true, returns samples and step between the consecutive numbers
dtype	Data type of output **ndarray**

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

Its output would be:

```
[10.   12.5   15.   17.5  20.]
```

```
# endpoint set to false
import numpy as np
x = np.linspace(10,20, 5, endpoint=False)
print x
```

The output would be:

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

```
# find retstep value
import numpy as np
x = np.linspace(1,2,5, retstep=True)
print x
# retstep here is 0.25
```

Now, the output would be:

```
(array([ 1.  ,  1.25,  1.5 ,  1.75,  2.  ]), 0.25)
```

numpy.logspace

- This function returns an **ndarray object that contains the numbers that are evenly spaced** on a log scale. Start and stop endpoints of the scale are indices of the base, usually 10.

numpy.logscale(start, stop, num, endpoint, base, dtype)

start	The starting point of the sequence is $base^{start}$
stop	The final value of sequence is $base^{stop}$
num	The number of values between the range. Default is 50
endpoint	If true, stop is the last value in the range
base	Base of log space, default is 10
dtype	Data type of output array. If not given, it depends upon other input arguments

```
import numpy as np
# default base is 10
a = np.logspace(1.0, 2.0, num=10)
print a
```

Its output would be as follows:

```
[ 10.          12.91549665    16.68100537    21.5443469     27.82559402
  35.93813664   46.41588834    59.94842503    77.42636827    100.        ]
```

```
# set base of log space to 2
import numpy as np
a = np.logspace(1,10,num=10, base=2)
print a
```

Now, the output would be:

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

Indexing & Slicing

```
import numpy as np
a = np.arange(10)
s = slice(2,7,2)
print a[s]
```

Its output is as follows:

```
[2  4  6]
```

```
import numpy as np
a = np.arange(10)
b = a[2:7:2]
print b
```

Here, we will get the same output:

```
[2  4  6]
```

```
# slice single item
import numpy as np
a = np.arange(10)
b = a[5]
print b
```

Its output is as follows:

```
5
```

```
# slice items starting from index
import numpy as np
a = np.arange(10)
print a[2:]
```

Now, the output would be:

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

```
# array to begin with
import numpy as np
a = np.array([[1,2,3],[3,4,5],[4,5,6]])
print 'Our array is:'
print a
print '\n'
```

```
Our array is:
[[1 2 3]
 [3 4 5]
 [4 5 6]]
```

```
# this returns array of items in the second column
print 'The items in the second column are:'
print a[...,1]
print '\n'
```

```
The items in the second column are:
[2 4 5]
```

```
# Now we will slice all items from the second row
print 'The items in the second row are:'
print a[1,...]
print '\n'
```

```
The items in the second row are:
[3 4 5]
```

Broadcasting

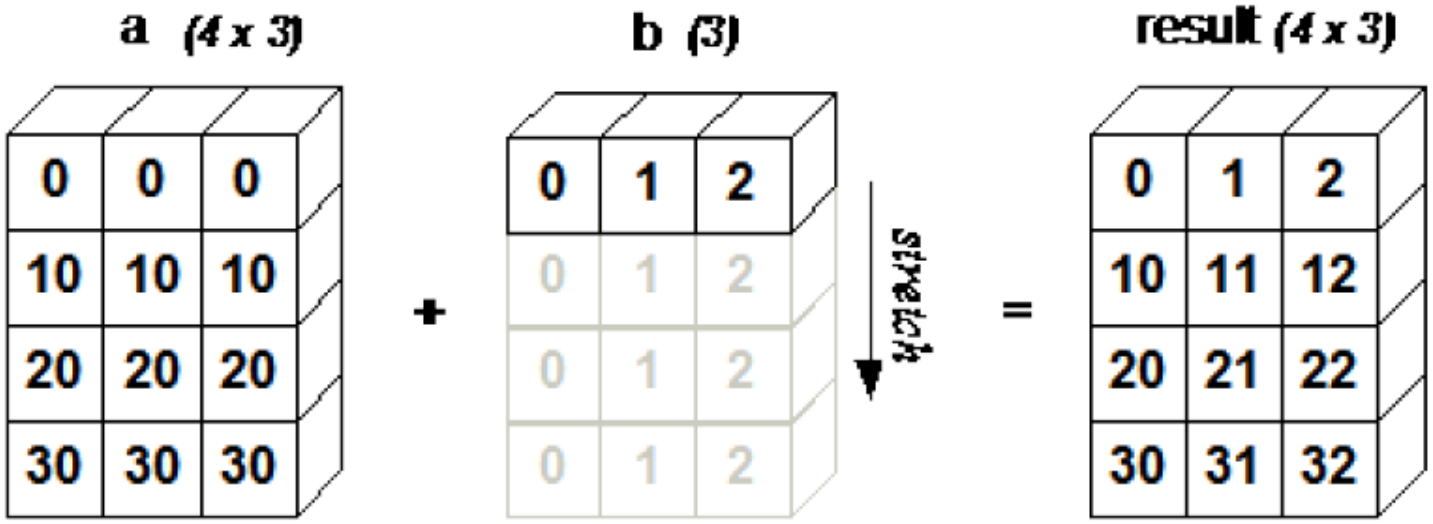
- Broadcasting is possible if the following rules are satisfied:
  - Array with smaller **ndim than the other is prepended with '1' in its shape.**
  - Size in each dimension of the output shape is maximum of the input sizes in that dimension.
  - An input can be used in calculation, if its size in a particular dimension matches the output size or its value is exactly 1.
  - If an input has a dimension size of 1, the first data entry in that dimension is used for all calculations along that dimension.

- A set of arrays is said to be **broadcastable if the above rules produce a valid result and** one of the following is true:
  - Arrays have exactly the same shape.
  - Arrays have the same number of dimensions and the length of each dimension is either a common length or 1.
  - Array having too few dimensions can have its shape prepended with a dimension of length 1, so that the above stated property is true.

```
import numpy as np
a = np.array([[ 0.0, 0.0, 0.0],[10.0,10.0,10.0],
              [20.0,20.0,20.0],[30.0,30.0,30.0]])
b = np.array([1.0,2.0,3.0])

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

print 'Second array:'
print b
print '\n'

print 'First Array + Second Array'
print a+b
```

```
First array:
[[  0.   0.   0.]
 [ 10.  10.  10.]
 [ 20.  20.  20.]
 [ 30.  30.  30.]]

Second array:
[ 1.  2.  3.]

First Array + Second Array
[[  1.   2.   3.]
 [ 11.  12.  13.]
 [ 21.  22.  23.]
 [ 31.  32.  33.]]
```

The following figure demonstrates how array **b** is broadcast to become compatible with **a**.



Iterating Over Array

- **numpy.nditer. It is an efficient multidimensional** iterator object using which it is possible to iterate over an array.

```
import numpy as np
a = np.arange(0,60,5)
a = a.reshape(3,4)
print 'Original array is:'
print a
print '\n'

print 'Modified array is:'
for x in np.nditer(a):
      print x,
```

```
Original array is:
[[ 0  5 10 15]
 [20 25 30 35]
 [40 45 50 55]]

Modified array is:
0 5 10 15 20 25 30 35 40 45 50 55
```

Modifying Array Values

- The **nditer object has another optional parameter called op_flags. Its default value is** read-only, but can be set to read-

```
import numpy as np
a = np.arange(0,60,5)
a = a.reshape(3,4)
print 'Original array is:'
print a
print '\n'

for x in np.nditer(a, op_flags=['readwrite']):
     x[...]=2*x

print 'Modified array is:'
print a
```

```
Original array is:
[[ 0  5 10 15]
 [20 25 30 35]
 [40 45 50 55]]

Modified array is:
[[  0  10  20  30]
 [ 40  50  60  70]
 [ 80  90 100 110]]
```

Statistical Functions

- **numpy.amin() and numpy.amax() -**These functions return the minimum and the maximum from the elements in the given array along the specified axis

```
a = np.array([[3,7,5],[8,4,3],[2,4,9]])
print 'Our array is:'
print a
print '\n'

print 'Applying amin() function:'
print np.amin(a,1)
print '\n'

print 'Applying amin() function again:'
print np.amin(a,0)
print '\n'

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

```
Our array is:
[[3 7 5]
 [8 4 3]
 [2 4 9]]
Applying amin() function:
[3 3 2]

Applying amin() function again:
[2 4 3]

Applying amax() function:
9
```

numpy.ptp()

- The **numpy.ptp() function returns the range (maximum-minimum) of values along an** axis.

```
import numpy as np
a = np.array([[3,7,5],[8,4,3],[2,4,9]])
print 'Our array is:'
print a
print '\n'

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

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

```
Our array is:
[[3 7 5]
 [8 4 3]
 [2 4 9]]

Applying ptp() function:
7

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

numpy.percentile()

- Percentile (or a centile) is a measure used in statistics indicating the value below which a given percentage of observations in a group of observations fall.

numpy.percentile(a, q, axis)

a	Input array
q	The percentile to compute must be between 0-100
axis	The axis along which the percentile is to be calculated

```
import numpy as np
a = np.array([[30,40,70],[80,20,10],[50,90,60]
print 'Our array is:'
print a
print '\n'

print 'Applying percentile() function:'
print np.percentile(a,50)
print '\n'

print 'Applying percentile() function along ax
print np.percentile(a,50, axis=1)
print '\n'

print 'Applying percentile() function along axis 0:'
print np.percentile(a,50, axis=0)
```

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

Applying percentile() function:
50.0

Applying percentile() function along axis 1:
[ 40.  20.  60.]

Applying percentile() function along axis 0:
[ 50.  40.  60.]
```

numpy.median()

- **Median is defined as the value separating the higher half of a data sample from the lower** half.

```
import numpy as np
a = np.array([[30,65,70],[80,95,10],[50,90
print 'Our array is:'
print a
print '\n'

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

print 'Applying median() function along ax
print np.median(a, axis=0)
print '\n'

print 'Applying median() function along ax
print np.median(a, axis=1)
```

```
Our array is:
[[30 65 70]
 [80 95 10]
 [50 90 60]]

Applying median() function:
65.0

Applying median() function along axis 0:
[ 50.  90.  60.]

Applying median() function along axis 1:
[ 65.  80.  60.]
```

numpy.mean()

```
import numpy as np
a = np.array([[1,2,3],[3,4,5],[4,5,6]])
print 'Our array is:'
print a
print '\n'

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

print 'Applying mean() function along axis 0:'
print np.mean(a, axis=0)
print '\n'

print 'Applying mean() function along axis 1:'
print np.mean(a, axis=1)
```

```
Our array is:
[[1 2 3]
 [3 4 5]
 [4 5 6]]

Applying mean() function:
3.66666666667

Applying mean() function along axis 0:
[ 2.66666667  3.66666667  4.66666667]

Applying mean() function along axis 1:
[ 2.  4.  5.]
```

Standard Deviation

std = sqrt(mean(abs(x - x.mean()))**2))

```
import numpy as np
print np.std([1,2,3,4])
```

It will produce the following output:

```
1.1180339887498949
```

Sort, Search & Counting Functions

- **numpy.sort()** - The sort() function returns a sorted copy of the input array.

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

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

```
import numpy as np
```

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

```
array([[3, 7],
       [9, 1]])
```

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

```
[[3 7]
 [1 9]]
```

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

```
[[3 1]
 [9 7]]
```

```
print (np.sort(a, axis=1))
```

```
[[3 7]
 [1 9]]
```

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

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

```
np.sort(x)
```

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

```
dt = np.dtype([('name', 'S10'),('age', int)])
a = np.array([("raju",21),("anil",25),("ravi", 17), ("amar",27)], dtype=dt)
a
```

```
array([(b'raju', 21), (b'anil', 25), (b'ravi', 17), (b'amar', 27)],
      dtype=[('name', 'S10'), ('age', '<i4')])
```

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

```
[(b'amar', 27) (b'anil', 25) (b'raju', 21) (b'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.

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

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

```
np.sort(x)
```

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

```
np.argsort(x)
```

```
array([1, 2, 0], dtype=int64)
```

```
y=np.argsort(x)
```

```
print (x[y])
```

```
[1 2 3]
```

```
for i in y: print (x[i])
```

```
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.

```
nm = ('raju','anil','ravi','amar')
dv = ('f.y.', 's.y.', 's.y.', 'f.y.')
```

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

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

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

```
['amar, f.y.', 'anil, s.y.', 'raju, f.y.', 'ravi, s.y.']
```

numpy.argmax() and numpy.argmin()

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

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

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

```
print (np.argmax(a))
```

```
7
```

```
print (a.flatten())
```

```
[30 40 70 80 20 10 50 90 60]
```

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

```
[1 2 0]
```

```
print(np.argmax(a, axis=1))
```

```
[2 0 1]
```

```
print(np.argmin(a))
```

```
5
```

```
print (a.flatten()[np.argmax(a, axis=1)])
```

```
[70 30 40]
```

numpy.nonzero()

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

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

```
array([[30, 40,  0],
       [ 0, 20, 10],
       [50,  0, 60]])
```

```
np.nonzero(a)
```

```
(array([0, 0, 1, 1, 2, 2], dtype=int64),
 array([0, 1, 1, 2, 0, 2], dtype=int64))
```

numpy.where()

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

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

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

```
y=np.where(x>3)
y
```

```
(array([1, 1, 2, 2, 2], dtype=int64), array([1, 2, 0, 1, 2], dtype=int64))
```

```
x[y]
```

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

numpy.extract()

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

```
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)
```

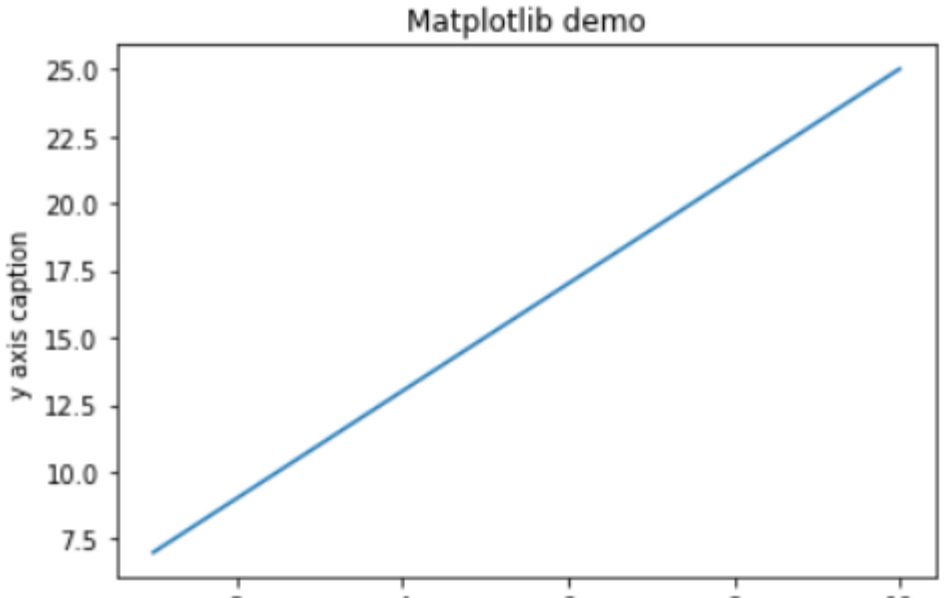
```
Our array is:
[[ 0.  1.  2.]
 [ 3.  4.  5.]
 [ 6.  7.  8.]]

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

Matplotlib

- Matplotlib is a plotting library for Python.

```
from matplotlib import pyplot as plt
```

```
x=np.arange(1,11)
y=2*x+5
plt.title("Matplotlib demo")
plt.xlabel("x axis caption")
plt.ylabel("y axis caption")
plt.plot(x,y)
plt.show()
```



I/O with NumPy

- **numpy.save()**

```
import numpy as np
a=np.array([1,2,3,4,5])
np.save('outfile',a)
```

To reconstruct array from **outfile.npy**, use **load()** function.

```
import numpy as np
b = np.load('outfile.npy')
print b
```

savetxt()

- The storage and retrieval of array data in simple text file format is done with **savetxt()** and **loadtxt() functions.**

```
import numpy as np
a = np.array([1,2,3,4,5])
np.savetxt('out.txt',a)
b = np.loadtxt('out.txt')
print b
```

It will produce the following output:

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

```
wines = np.genfromtxt("wine_sample.csv", delimiter=",")
wines
```

```
array([[  7.,   1.,   0.,   2.,   0.,  11.,  34.,   1.,   4.,   1.,   9.,
          5.],
       [  8.,   1.,   0.,   3.,   0.,  25.,  67.,   1.,   3.,   1.,  10.,
          5.],
       [  8.,   1.,   0.,   2.,   0.,  15.,  54.,   1.,   3.,   1.,  10.,
          5.],
       [ 11.,   0.,   1.,   2.,   0.,  17.,  60.,   1.,   3.,   1.,  10.,
          6.],
       [  7.,   1.,   0.,   2.,   0.,  11.,  34.,   1.,   4.,   1.,   9.,
          5.]])
```

```
a=np.asarray(wines,dtype=int)
```

```
a
```

```
array([[ 7,  1,  0,  2,  0, 11, 34,  1,  4,  1,  9,  5],
       [ 8,  1,  0,  3,  0, 25, 67,  1,  3,  1, 10,  5],
       [ 8,  1,  0,  2,  0, 15, 54,  1,  3,  1, 10,  5],
       [11,  0,  1,  2,  0, 17, 60,  1,  3,  1, 10,  6],
       [ 7,  1,  0,  2,  0, 11, 34,  1,  4,  1,  9,  5]])
```

Array Operation

- **numpy.append**

numpy.append(arr, values, axis)

- **numpy.insert**

numpy.insert(arr, obj, values, axis)

- **numpy.delete**

Numpy.delete(arr, obj, axis)

- **numpy.unique**

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

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

```
print (np.append(a, [7,8,9]))
```

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

```
print (np.append(a, [[7,8,9]],axis=0))
```

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

```
print (np.append(a, [[5,5,5],[7,8,9]],axis=1))
```

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

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

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

```
print (np.insert(a,2,[11,12]))
```

```
[ 1  2 11 12  3  4  5  6]
```

```
print (np.insert(a,1,[11],axis=0))
```

```
[[ 1  2]
 [11 11]
 [ 3  4]
 [ 5  6]]
```

```
print (np.insert(a,1,11,axis=1))
```

```
[[ 1 11  2]
 [ 3 11  4]
 [ 5 11  6]]
```

```
a = np.arange(12).reshape(3,4)
a
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
print (np.delete(a,5))
```

```
[ 0  1  2  3  4  6  7  8  9 10 11]
```

```
print (np.delete(a,1,axis=1))
```

```
[[ 0  2  3]
 [ 4  6  7]
 [ 8 10 11]]
```

```
a = np.array([1,2,3,4,5,6,7,8,9,10])
a
```

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

```
print (np.delete(a, np.s_[::2]))
```

```
[ 2  4  6  8 10]
```