To know about Numpy check my blog (http://www.bigdataexaminer.com/5-amazingly-powerful-python-libraries-for-data-science/)

N- Dimensional array

Arrays allows you to perform mathematical operations on whole blocks of data.

In [1]:
```
# easiest way to create an array is by using an array function
import numpy as np # I am importing numpy as np

scores = [89,56.34, 76,89, 98]
first_arr =np.array(scores)
print first_arr
print first_arr.dtype  # .dtype return the data type of the array object
```

```
[ 89.    56.34  76.    89.    98.  ]
float64
```

In [22]:
```
# Nested lists with equal length, will be converted into a multidimension
scores_1 = [[34,56,23,89], [11,45,76,34]]
second_arr = np.array(scores_1)
print second_arr
print second_arr.ndim  #.ndim gives you the dimensions of an array.
print second_arr.shape #(number of rows, number of columns)
print second_arr.dtype
```

```
[[34 56 23 89]
 [11 45 76 34]]
2
(2L, 4L)
int32
```

In [28]:
```
x = np.zeros(10)  # returns a array of zeros, the same applies for np.ones
x
```

Out[28]:
```
array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.])
```

In [30]:
```
np.zeros((4,3)) # you can also mention the shape of the array
```

Out[30]:
```
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.],
       [ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
```

In [34]:
```
np.arange(15)
```

Out[34]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

In [36]:
```
np.eye(6) # Create a square N x N identity matrix (1's on the diagonal an
```

Out[36]:
```
array([[ 1.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  1.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  1.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  1.]])
```

In [10]:
```
#Batch operations on data can be performed without using for loops, this
scores = [89,56.34, 76,89, 98]
first_arr =np.array(scores)
print first_arr
print first_arr * first_arr
print first_arr - first_arr
print 1/(first_arr)
print first_arr ** 0.5
```

```
[ 89.    56.34  76.    89.    98.  ]
[ 7921.      3174.1956  5776.      7921.      9604.    ]
[ 0.  0.  0.  0.  0.]
[ 0.01123596  0.01774938  0.01315789  0.01123596  0.01020408]
[ 9.43398113  7.5059976   8.71779789  9.43398113  9.89949494]
```

Indexing and Slicing

In [26]:
```
# you may want to select a subset of your data, for which Numpy array ind
new_arr = np.arange(12)
print new_arr
print new_arr[5]
print new_arr[4:9]
new_arr[4:9] = 99 #assign sequence of values from 4 to 9 as 99
print new_arr
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11]
5
[4 5 6 7 8]
[ 0  1  2  3 99 99 99 99 99  9 10 11]
```

In [27]:
```
# A major diffence between lists and array is that, array slices are view
# the data is not copied, and any modifications to the view will be refle
#  array.
modi_arr = new_arr[4:9]
modi_arr[1] = 123456
print new_arr                  # you can see the changes are refelected i
modi_arr[:]                   # the sliced variable
```

```
[     0      1      2      3     99 123456     99     99     99      9
     10     11]
```

Out[27]:
```
array([    99, 123456,     99,     99,     99])
```

In [9]:

```
# arrays can be treated like matrices
matrix_arr =np.array([[3,4,5],[6,7,8],[9,5,1]])
print matrix_arr
print matrix_arr[1]
print matrix_arr[0][2] #first row and third column
print matrix_arr[0,2] # This is same as the above operation

from IPython.display import Image  # importing a image from my computer.
i = Image(filename='Capture.png')
i # Blue print of a matrix
```

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

Out[9]:

	Column 0	Column 1	Column 2
Row 0	0,0	0,1	0,2
Row 1	1,0	1,1	1,2
Row 2	2,0	2,1	2,2

In [8]:

```
cd C:\Users\tk\Desktop\pics # changing my directory
```

```
C:\Users\tk\Desktop\pics
```

In [37]:

```
# 3d arrays -> this is a 2x2x3 array
three_d_arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]
print three_d_arr
print "returns the second list inside first list {}".format(three_d_arr[0
```

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

 [[ 7  8  9]
  [10 11 12]]]
returns the second list inside first list [4 5 6]
```

In [39]:

```
three_d_arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]
print three_d_arr[0]
#if you omit later indices, the returned object will be a lowerdimensiona
# ndarray consisting of all the data along the higher dimensions
```

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

I have used format (https://docs.python.org/2/tutorial/inputoutput.html) function in the below cell.

In [62]:
```
copied_values = three_d_arr[0].copy() # copy arr[0] value to copied_value
three_d_arr[0]= 99  # change all values of arr[0] to 99
print "New value of three_d_arr: {}".format(three_d_arr)  # check the new
three_d_arr[0] = copied_values # assign copied values back to three_d_arr
print" three_d_arr again: {}".format(three_d_arr)
```

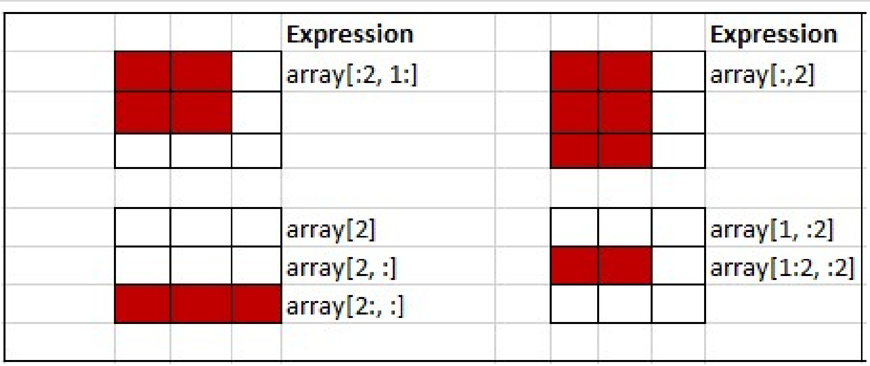
```
New value of three_d_arr: [[[99 99 99]
  [99 99 99]]

 [[ 7  8  9]
  [10 11 12]]]
 three_d_arr again: [[[99 99 99]
  [99 99 99]]

 [[ 7  8  9]
  [10 11 12]]]
```

In [76]:
```
matrix_arr =np.array([[3,4,5],[6,7,8],[9,5,1]])
print "The original matrix {}:".format(matrix_arr)
print "slices the first two rows:{}".format(matrix_arr[:2]) # similar to
print "Slices the first two rows and two columns:{}".format(matrix_arr[:2
print "returns 6 and 7: {}".format(matrix_arr[1,:2])
print "Returns first column: {}".format(matrix_arr[:,:1]) #Note that a co
```

```
The original matrix [[3 4 5]
 [6 7 8]
 [9 5 1]]:
slices the first two rows:[[3 4 5]
 [6 7 8]]
Slices the first two rows and two columns:[[4 5]
 [7 8]]
returns 6 and 7: [6 7]
Returns first column: [[3]
 [6]
 [9]]
```

In [80]:
```
from IPython.display import Image  # importing a image from my computer.
j = Image(filename='Expre.png')
j # diagrammatic explanation of matrix array slicing works.
```

Out[80]:



In [4]:
```
#Import random module from Numpy
personals = np.array(['Manu', 'Jeevan', 'Prakash', 'Manu', 'Prakash', 'Je
print personals == 'Manu' #checks for the string 'Manu' in personals. If
```

```
[ True False False  True False False False]
```

In [5]:
```
from numpy import random
random_no = random.randn(7,4)
print random_no
random_no[personals =='Manu'] #The function returns the rows for which th
# Check the image displayed in the cell below.
```

```
[[-0.129557    0.3684001  -0.15747451 -0.1196816 ]
 [-0.35946571 -1.23477985  1.08186057 -0.61596683]
 [ 1.67096505  1.11183755 -0.39640455  0.22848279]
 [-0.27989438 -1.51275966 -0.48825407  1.32425359]
 [-0.04493194 -1.10371501 -0.52742166 -1.06265549]
 [ 1.16938298 -0.60478133  1.40615125 -1.35350336]
 [ 0.86325448  1.97577081  0.05339779  0.71515521]]
```

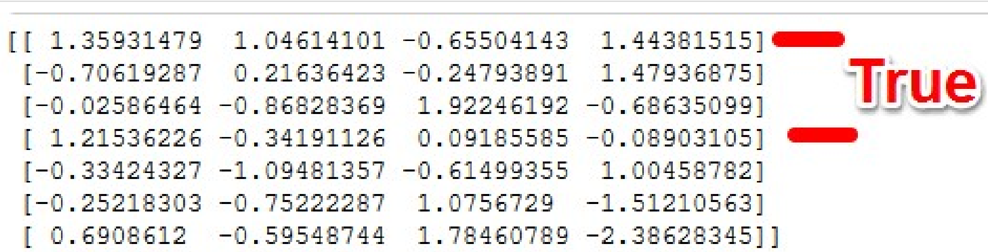
Out[5]:
```
array([[-0.129557  ,  0.3684001 , -0.15747451, -0.1196816 ],
       [-0.27989438, -1.51275966, -0.48825407,  1.32425359]])
```

In [10]:
```
cd C:\Users\Manu\Desktop
```

```
C:\Users\Manu\Desktop
```

In [11]:
```
from IPython.display import Image
k = Image(filename='Matrix.png')
k
```

Out[11]:



```
[[ 1.35931479  1.04614101 -0.65504143  1.44381515]
 [-0.70619287  0.21636423 -0.24793891  1.47936875]
 [-0.02586464 -0.86828369  1.92246192 -0.68635099]
 [ 1.21536226 -0.34191126  0.09185585 -0.08903105]
 [-0.33424327 -1.09481357 -0.61499355  1.00458782]
 [-0.25218303 -0.75222287  1.0756729  -1.51210563]
 [ 0.6908612  -0.59548744  1.78460789 -2.38628345]]
```


In [12]:
```
random_no[personals == 'Manu', 2:] #First two columns and first two rows.
```

Out[12]:
```
array([[-0.15747451, -0.1196816 ],
       [-0.48825407,  1.32425359]])
```

In [13]:
```
# To select everything except 'Manu', you can != or negate the condition
print personals != 'Manu'
random_no[-(personals == 'Manu')] #get everything except 1st and 4th rows
```

```
[False  True  True False  True  True  True]
```

Out[13]:
```
array([[-0.35946571, -1.23477985,  1.08186057, -0.61596683],
       [ 1.67096505,  1.11183755, -0.39640455,  0.22848279],
       [-0.04493194, -1.10371501, -0.52742166, -1.06265549],
       [ 1.16938298, -0.60478133,  1.40615125, -1.35350336],
       [ 0.86325448,  1.97577081,  0.05339779,  0.71515521]])
```

In [18]:
```
# you can use boolean operator &(and), |(or)
new_variable = (personals == 'Manu') | (personals == 'Jeevan')
print new_variable
random_no[new_variable]
```

```
[ True  True False  True False  True False]
```

Out[18]:
```
array([[-0.129557  ,  0.3684001 , -0.15747451, -0.1196816 ],
       [-0.35946571, -1.23477985,  1.08186057, -0.61596683],
       [-0.27989438, -1.51275966, -0.48825407,  1.32425359],
       [ 1.16938298, -0.60478133,  1.40615125, -1.35350336]])
```

In [22]:
```
random_no[random_no < 0] =0
random_no # This will set all negative values to zero
```

Out[22]:
```
array([[ 0.        ,  0.3684001 ,  0.        ,  0.        ],
       [ 0.        ,  0.        ,  1.08186057,  0.        ],
       [ 1.67096505,  1.11183755,  0.        ,  0.22848279],
       [ 0.        ,  0.        ,  0.        ,  1.32425359],
       [ 0.        ,  0.        ,  0.        ,  0.        ],
       [ 1.16938298,  0.        ,  1.40615125,  0.        ],
       [ 0.86325448,  1.97577081,  0.05339779,  0.71515521]])
```

In [24]:
```
random_no[ personals != 'Manu'] = 9 # This will set all rows except 1 and
random_no
```

Out[24]:
```
array([[ 0.        ,  0.3684001 ,  0.        ,  0.        ],
       [ 9.        ,  9.        ,  9.        ,  9.        ],
       [ 9.        ,  9.        ,  9.        ,  9.        ],
       [ 0.        ,  0.        ,  0.        ,  1.32425359],
       [ 9.        ,  9.        ,  9.        ,  9.        ],
       [ 9.        ,  9.        ,  9.        ,  9.        ],
       [ 9.        ,  9.        ,  9.        ,  9.        ]])
```

Fancy Indexing(Indexing using integer arrays)

Fancy indexing copies data into a new array

In [19]:
```
from numpy import random
algebra = random.randn(7,4)  # empty will return a matrix of size 7,4
for j in range(7):
    algebra[j] = j
algebra
```

Out[19]:
```
array([[ 0.,  0.,  0.,  0.],
       [ 1.,  1.,  1.,  1.],
       [ 2.,  2.,  2.,  2.],
       [ 3.,  3.,  3.,  3.],
       [ 4.,  4.,  4.,  4.],
       [ 5.,  5.,  5.,  5.],
       [ 6.,  6.,  6.,  6.]])
```

In [23]:
```
# To select a subset of rows in particular order, you can simply pass a l
algebra[[4,5,1]] #returns a subset of rows
```

Out[23]:
```
array([[ 4.,  4.,  4.,  4.],
       [ 5.,  5.,  5.,  5.],
       [ 1.,  1.,  1.,  1.]])
```

In [33]:
```
fancy = np.arange(36).reshape(9,4) #reshape is to reshape an array
print fancy
fancy[[1,4,3,2],[3,2,1,0]] #the position of the output array are[(1,3),(4
```

```
[[ 0  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]]
```

Out[33]:
```
array([ 7, 18, 13,  8])
```

In [39]:
```
fancy[[1, 4, 8, 2]][:, [0, 3, 1, 2]] # entire first row is selected, but
```

Out[39]:
```
array([[ 4,  7,  5,  6],
       [16, 19, 17, 18],
       [32, 35, 33, 34],
       [ 8, 11,  9, 10]])
```

In [42]:
```
# another way to do the above operation is by using np.ix_ function.
fancy[np.ix_([1,4,8,2],[0,3,1,2])]
```

Out[42]:
```
array([[ 4,  7,  5,  6],
       [16, 19, 17, 18],
       [32, 35, 33, 34],
       [ 8, 11,  9, 10]])
```

Transposing Arrays

In [47]:
```
transpose= np.arange(12).reshape(3,4)
transpose.T # the shape has changed to 4,3
```

Out[47]:
```
array([[ 0,  4,  8],
       [ 1,  5,  9],
       [ 2,  6, 10],
       [ 3,  7, 11]])
```

In [48]:
```
#you can use np.dot function to perform matrix computations. You can calc
np.dot(transpose.T, transpose)
```

Out[48]:
```
array([[ 80,  92, 104, 116],
       [ 92, 107, 122, 137],
       [104, 122, 140, 158],
       [116, 137, 158, 179]])
```

universal functions

They perform element wise operations on data in arrays.

In [53]:

```
funky =np.arange(8)
print np.sqrt(funky)
print np.exp(funky) #exponent of the array
# these are called as unary functions
```

```
[ 0.          1.          1.41421356  1.73205081  2.          2.2360679
  2.44948974  2.64575131]
[  1.00000000e+00   2.71828183e+00   7.38905610e+00   2.00855369e+01
   5.45981500e+01   1.48413159e+02   4.03428793e+02   1.09663316e+03]
```

In [62]:

```
# Binary functions take two value, Others such as maximum, add
x = random.randn(10)
y = random.randn(10)
print x
print y
print np.maximum(x,y)# element wise operation
print np.modf(x)# function modf returns the fractional and integral parts
```

```
[-0.47538326 -0.32308133  1.45505923 -0.53196376 -1.34427866 -2.1440955
 -0.96296558  0.14068437 -0.29208196 -1.17537313]
[-1.68868842 -0.53788536 -1.01887225 -0.02972594 -1.04607062 -2.0863616
  0.34398903 -0.64183089  1.55401001  0.73270627]
[-0.47538326 -0.32308133  1.45505923 -0.02972594 -1.04607062 -2.0863616
  0.34398903  0.14068437  1.55401001  0.73270627]
(array([-0.47538326, -0.32308133,  0.45505923, -0.53196376, -0.34427866
       -0.14409558, -0.96296558,  0.14068437, -0.29208196, -0.17537313]
```

In [67]:

```
# List of unary functions avaliable
from IPython.display import Image
l = Image(filename='unary functions.png')
l
```

Out[67]:

Function	Description
abs, fabs	Compute the absolute value element-wise for integer, floating point, or complex values. Use fabs as a faster alternative for non-complex-valued data
sqrt	Compute the square root of each element. Equivalent to arr ** 0.5
square	Compute the square of each element. Equivalent to arr ** 2
exp	Compute the exponent e^x of each element
log, log10, log2, log1p	Natural logarithm (base *e*), log base 10, log base 2, and log(1 + x), respectively
sign	Compute the sign of each element: 1 (positive), 0 (zero), or -1 (negative)
ceil	Compute the ceiling of each element, i.e. the smallest integer greater than or equal to each element
floor	Compute the floor of each element, i.e. the largest integer less than or equal to each element
rint	Round elements to the nearest integer, preserving the dtype
modf	Return fractional and integral parts of array as separate array
isnan	Return boolean array indicating whether each value is NaN (Not a Number)
isfinite, isinf	Return boolean array indicating whether each element is finite (non-inf, non-NaN) or infinite, respectively
cos, cosh, sin, sinh, tan, tanh	Regular and hyperbolic trigonometric functions
arccos, arccosh, arcsin, arcsinh, arctan, arctanh	Inverse trigonometric functions
logical_not	Compute truth value of not x element-wise. Equivalent to -arr.

In [69]:

```
#List of binary functions available
from IPython.display import Image
l = Image(filename='binary functions.png')
l
#logical operators , and  greater, greater_equal,less, less_equal, equal,
```

Out[69]:

Function	Description
add	Add corresponding elements in arrays
subtract	Subtract elements in second array from first array
multiply	Multiply array elements
divide, floor_divide	Divide or floor divide (truncating the remainder)
power	Raise elements in first array to powers indicated in second array
maximum, fmax	Element-wise maximum. fmax ignores NaN
minimum, fmin	Element-wise minimum. fmin ignores NaN
mod	Element-wise modulus (remainder of division)
copysign	Copy sign of values in second argument to values in first argument

Data processing using Arrays

In [86]:
```
mtrices =np.arange(-5,5,1)
x, y = np.meshgrid(mtrices, mtrices) #mesh grid function takes two 1 d ar
print "Matrix values of y: {}".format(y)
print "Matrix values of x: {}".format(x)
```

```
Matrix values of y: [[-5 -5 -5 -5 -5 -5 -5 -5 -5 -5]
 [-4 -4 -4 -4 -4 -4 -4 -4 -4 -4]
 [-3 -3 -3 -3 -3 -3 -3 -3 -3 -3]
 [-2 -2 -2 -2 -2 -2 -2 -2 -2 -2]
 [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
 [ 0  0  0  0  0  0  0  0  0  0]
 [ 1  1  1  1  1  1  1  1  1  1]
 [ 2  2  2  2  2  2  2  2  2  2]
 [ 3  3  3  3  3  3  3  3  3  3]
 [ 4  4  4  4  4  4  4  4  4  4]]
Matrix values of x: [[-5 -4 -3 -2 -1  0  1  2  3  4]
 [-5 -4 -3 -2 -1  0  1  2  3  4]
 [-5 -4 -3 -2 -1  0  1  2  3  4]
 [-5 -4 -3 -2 -1  0  1  2  3  4]
 [-5 -4 -3 -2 -1  0  1  2  3  4]
 [-5 -4 -3 -2 -1  0  1  2  3  4]
 [-5 -4 -3 -2 -1  0  1  2  3  4]
 [-5 -4 -3 -2 -1  0  1  2  3  4]
 [-5 -4 -3 -2 -1  0  1  2  3  4]
 [-5 -4 -3 -2 -1  0  1  2  3  4]]
```

zip (http://stackoverflow.com/questions/13704860/zip-lists-in-python) function is clearly explained here.

In [124]:
```
x1= np.array([1,2,3,4,5])
y1 = np.array([6,7,8,9,10])
cond =[True, False, True, True, False]
#If you want to take a value from x1 whenever the corresponding value in
z1 = [(x,y,z) for x,y,z in zip(x1, y1, cond)] # I have used zip function
print z1
np.where(cond, x1, y1)
```

```
[(1, 6, True), (2, 7, False), (3, 8, True), (4, 9, True), (5, 10, False
```

Out[124]:
```
array([ 1,  7,  3,  4, 10])
```

In [132]:
```
ra = np.random.randn(5,5)
# If you want to replace negative values in ra with -1 and positive value
print ra
print np.where(ra>0, 1, -1) # If values in ra are greater than zero, repl
# to set only positive values
np.where(ra >0, 1, ra) # same implies to negative values
```

```
[[-0.91593384  0.38253326 -0.13340929 -0.12353528 -0.90849552]
 [ 2.23109011 -0.7980066   0.13600282 -0.2447923   1.32865533]
 [-0.65568719 -1.48154609  0.8033841  -0.84157511 -0.19588005]
 [ 1.42527047  0.63082249 -0.80092209 -0.69935209  0.20470869]
 [ 0.18245815 -0.99953295  0.05586992  0.38031972  0.60522581]]
[[-1  1 -1 -1 -1]
 [ 1 -1  1 -1  1]
 [-1 -1  1 -1 -1]
 [ 1  1 -1 -1  1]
 [ 1 -1  1  1  1]]
```

Out[132]:
```
array([[-0.91593384,  1.        , -0.13340929, -0.12353528, -0.90849552
       [ 1.        , -0.7980066 ,  1.        , -0.2447923 ,  1.
       [-0.65568719, -1.48154609,  1.        , -0.84157511, -0.19588005
       [ 1.        ,  1.        , -0.80092209, -0.69935209,  1.
       [ 1.        , -0.99953295,  1.        ,  1.        ,  1.
```

Statistical methods

In [136]:
```
thie = np.random.randn(5,5)
print thie.mean() # calculates the mean of thie
print np.mean(thie) # alternate method to calculate mean
print thie.sum()
```

```
0.286291297223
0.286291297223
7.15728243058
```

In [36]:
```
jp =np.arange(12).reshape(4,3)
print"The arrays are: {}".format(jp)
print "The sum of rows are :{}".format(np.sum(jp, axis =0)) #axis =0, giv
# remember this zero is for columns and one is for rows.
```

```
The arrays are: [[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
The sum of rows are :[18 22 26]
```

In [35]:
```
print jp.sum(1)#returns sum of rows
```

```
[ 3 12 21 30]
```

In [37]:
```
jp.cumsum(0)  #cumulative sum of columns, try the same for jp.cumprod(0)
```

Out[37]:
```
array([[ 0,  1,  2],
       [ 3,  5,  7],
       [ 9, 12, 15],
       [18, 22, 26]])
```

In [38]:
```
jp.cumsum(1)#cumulative sum of rows
```

Out[38]:
```
array([[ 0,  1,  3],
       [ 3,  7, 12],
       [ 6, 13, 21],
       [ 9, 19, 30]])
```

In [47]:
```
xp =np.random.randn(100)
print(xp > 0).sum() # sum of all positive values
print (xp < 0).sum()
tandf =np.array([True,False,True,False,True,False])
print tandf.any()#checks if any of the values are true
print tandf.all() #returns false even if a single value is false
#These methods also work with non-boolean arrays, where non-zero elements
```

```
45
55
True
False
```

Other array functions are:

std, var -> standard deviation and variance

min, max -> Minimum and Maximum

argmin, argmax -> Indices of minimum and maximum elements

Sorting

In [55]:
```
lp = np.random.randn(8)
print lp
lp.sort()
lp
```

```
[-0.38465299 -0.84381465 -1.78393531 -0.80242681 -2.54136215 -0.4735474
 -1.17517075  0.23759082]
```

Out[55]:
```
array([-2.54136215, -1.78393531, -1.17517075, -0.84381465, -0.80242681,
       -0.47354742, -0.38465299,  0.23759082])
```

In [57]:
```
tp = np.random.randn(4,4)
tp
```

Out[57]:
```
array([[ 0.4968525 , -0.65497365, -0.43687651,  0.51706412],
       [-1.39148137, -0.0166924 , -0.82572908,  2.20839298],
       [-0.5400157 , -0.8311936 , -1.92611011,  0.04556166],
       [ 0.41679611, -1.1659837 , -1.7181857 ,  0.15529182]])
```

In [60]:
```
tp.sort(1) #check the rows are sorted
tp
```

Out[60]:
```
array([[-0.65497365, -0.43687651,  0.4968525 ,  0.51706412],
       [-1.39148137, -0.82572908, -0.0166924 ,  2.20839298],
       [-1.92611011, -0.8311936 , -0.5400157 ,  0.04556166],
       [-1.7181857 , -1.1659837 ,  0.15529182,  0.41679611]])
```

In [61]:
```
personals = np.array(['Manu', 'Jeevan', 'Prakash', 'Manu', 'Prakash', 'Je
np.unique(personals)# returns the unique elements in the array
```

Out[61]:
```
array(['Jeevan', 'Manu', 'Prakash'],
      dtype='|S7')
```

In [65]:
```
set(personals) # set is an alternative to unique function
```

Out[65]:
```
{'Jeevan', 'Manu', 'Prakash'}
```

In [67]:
```
np.in1d(personals, ['Manu']) #in1d function checks for the value 'Manu' a
```

Out[67]:
```
array([ True, False, False,  True, False, False, False], dtype=bool)
```

Other Functions are :

intersect1d(x, y)-> Compute the sorted, common elements in x and y
union1d(x,y) -> compute the sorted union of elements
setdiff1d(x,y) -> set difference, elements in x that are not in y
setxor1d(x, y) -> Set symmetric differences; elements that are in either of the arrays, but not both

Linear Algebra

In [75]:
```
cp = np.array([[1,2,3],[4,5,6]])
dp = np.array([[7,8],[9,10],[11,12]])
print "CP array :{}".format(cp)
print "DP array :{}".format(dp)
```

```
CP array :[[1 2 3]
 [4 5 6]]
DP array :[[ 7  8]
 [ 9 10]
 [11 12]]
```

In [73]:
```
# element wise multiplication
cp.dot(dp) # this is equivalent to np.dot(x,y)
```

Out[73]:
```
array([[ 58,  64],
       [139, 154]])
```

In [77]:
```
np.dot(cp, np.ones(3))
```

Out[77]:
```
array([  6.,  15.])
```

In [84]:
```
# numpy.linalg has standard matrix operations like determinants and inver
from numpy.linalg import inv, qr
cp = np.array([[1,2,3],[4,5,6]])
new_mat = cp.T.dot(cp) # multiply cp inverse and cp, this is element wise
print new_mat
```

```
[[17 22 27]
 [22 29 36]
 [27 36 45]]
```

In [90]:
```
sp = np.random.randn(5,5)
print inv(sp)
rt = inv(sp)
```

```
[[  8.42073934  -3.99404791  -1.02750024  -9.15141449 -11.83177632]
 [  0.99455489   0.12614541   0.97324631   0.13731371   1.83602625]
 [  7.22433965  -3.9236319   -1.72053933  -8.26352406 -11.80445805]
 [  4.35711911  -2.62701594  -0.75752399  -4.80133342  -6.89057351]
 [  4.97536913  -1.66709125   0.42132364  -4.00769704  -4.45711904]]
```

In [91]:
```
# to calculate the product of a matrix and its inverse
sp.dot(rt)
```

Out[91]:
```
array([[  1.00000000e+00,  -6.66133815e-16,  -3.88578059e-16,
         -4.44089210e-16,  -5.77315973e-15],
       [ -8.88178420e-16,   1.00000000e+00,   1.11022302e-16,
          4.44089210e-16,   8.88178420e-16],
       [ -2.66453526e-15,   2.22044605e-16,   1.00000000e+00,
         -3.55271368e-15,   2.22044605e-15],
       [  8.88178420e-16,   0.00000000e+00,  -1.11022302e-16,
          1.00000000e+00,  -8.88178420e-16],
       [  0.00000000e+00,  -6.66133815e-16,   1.66533454e-16,
          8.88178420e-16,   1.00000000e+00]])
```

In [95]:
```
q,r = qr(sp)
print q
r
```

```
[[-0.50510571  0.0181599   0.07531349  0.59150958 -0.62368481]
 [ 0.13921471 -0.40513763  0.84451738  0.24413444  0.20897736]
 [ 0.53635022 -0.51829708 -0.46188958  0.47703793 -0.05281481]
 [-0.66103319 -0.49468555 -0.25644088  0.01307464  0.50238278]
 [ 0.02917284  0.56761612 -0.04488163  0.6023111   0.55871984]]
```

Out[95]:
```
array([[ 2.90927288, -0.76452754, -3.08539037,  0.77536573, -1.07156322
       [-0.        ,  2.28961296,  1.31005059, -0.44393071, -1.96748764
       [-0.        ,  0.        ,  1.48340931, -2.65558951,  0.18679631
       [ 0.        ,  0.        ,  0.        , -0.37900614,  0.4507976
       [-0.        , -0.        , -0.        , -0.        , -0.12535448
```

Other Matrix Functions

diag : Return the diagonal (or off-diagonal) elements of a square matrix as a 1D array, or convert a 1D array into a square
matrix with zeros on the off-diagonal

trace: Compute the sum of the diagonal elements

det: Compute the matrix determinant

eig: Compute the eigenvalues and eigenvectors of a square matrix

pinv: Compute the pseudo-inverse of a square matrix

svd: Compute the singular value decomposition (SVD)

solve: Solve the linear system Ax = b for x, where A is a square matrix

lstsq: Compute the least-squares solution to y = Xb