**ASSIGNMENT 1**

**Introduction**

First we downloaded PyCharm application to run python programs. We created a project named Example1.This Project is present in Cdrive. Then we created a new file by clicking on new->file and saved as name.py(sample1.py).

Then we used dependency packages (in c language they are called as library Functions).We installed the Dependency packages from settings.We installed numpy from there

**NUMPY**

Numpy stands for numerical python. Numpy is the dependency package that we installed. Numpy is used for fast and efficient operation on an array. It is a multidimensional array.

CODE

**import** numpy **as** np  
my\_list1=[1,2,3,4]  
my\_list2=[5,6,7,8]  
my\_array=np.array([my\_list1,my\_list2])  
**print** my\_array  
**print** my\_array.shape  
my\_array.dtype

OUTPUT

[[1 2 3 4]

[5 6 7 8]]

(2L, 4L)

The next we did was using functions zeros ,ones, empty ,eye, arrange. In zeros it creates a new arraywith (1\*n) elements.All elements are zero.

The ones create a multidimensional array. All elements of that array are 1.

The empty is similar as zeros.

The eye function is identity matrix. Diagonal elements as 1.

CODE

**import** numpy **as** np  
my\_list1=[1,2,3,4]  
my\_list2=[5,6,7,8]  
my\_array=np.array([my\_list1,my\_list2])

new\_array1=np.zeros(5)  
**print** new\_array1  
new\_array1=np.ones([5,5])  
**print** new\_array1  
new\_array1=np.empty(5)  
**print** new\_array1  
new\_array1=np.eye(5)  
**print** new\_array1  
new\_array1=np.arange(5,30,3)  
**print** new\_array1

OUTPUT

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

[[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.]]

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

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

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

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

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

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

[ 5 8 11 14 17 20 23 26 29]

**SCALAR OPERATIONS ON NUMPY**

If we run the following code

Import numpy as np

Print 5/2

We get output as 2 as it does integer division. To get 2.5 we need to do the following .

**from** \_\_future\_\_ **import** division  
**import** numpy **as** np

print 5/2

we get 2.5

following are the scalar operations on array.

CODE

**from** \_\_future\_\_ **import** division  
**import** numpy **as** np  
array1=np.array([[1,2,3,4],[5,6,7,8]])  
**print** array1  
*#multiplication*array2=array1\*array1  
**print** array2  
array3=array1\*\*3  
**print** array3  
array4=array1-array1  
**print** array4  
array5=array2-array1  
**print** array5  
**print** 1/array1

OUTPUT

[[1 2 3 4]

[5 6 7 8]]

[[ 1 4 9 16]

[25 36 49 64]]

[[ 1 8 27 64]

[125 216 343 512]]

[[0 0 0 0]

[0 0 0 0]]

[[ 0 2 6 12]

[20 30 42 56]]

[[1. 0.5 0.33333333 0.25 ]

[0.2 0.16666667 0.14285714 0.125 ]]

**Indexing the array(1 dimensional)**

We learnt how to access members of the array using indexes.

CODE

**import** numpy **as** np  
arr = np.arange(0,12)  
**print** arr  
  
**print** arr[0]  
**print** arr[0:5]  
arr[0:5]=20  
**print** arr  
arr2= arr[0:6]  
  
arr2[:]=29  
**print** arr

OUTPUT

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

0

[0 1 2 3 4]

[20 20 20 20 20 5 6 7 8 9 10 11]

[29 29 29 29 29 29 6 7 8 9 10 11]

Here arr2 is created for arr there is no separate memoy allocated to it . Numpy works to minimize memory problem.

To effect on previous array(new array copy)

array=arr.copy()

**array indexing (2 dimensional)**

slice is a part of the original 2D array.

CODE

**import** numpy **as** np  
arr2d=np.array([[1,2,3],[4,5,6],[7,8,9]])  
**print** arr2d  
**print** arr2d[0]  
**print** arr2d[1][2]  
  
slice1=arr2d[0:1,0:2]  
**print** slice1  
slice2=arr2d[:2,1:]  
**print** slice2  
arr2d[:2,1:]=15  
**print** arr2d

#using loops to index  
arr\_length=arr2d.shape[0]  
**for** i **in** range(arr\_length):  
 arr2d[i]=i;  
**print** arr2d;

#one more way of accessing rows  
**print** arr2d[[0,1]]  
**print** arr2d[[1,0]]

Here i manages values of all columns of ith row (all columns value=i for ith row)

OUTPUT

[[1 2 3]

[4 5 6]

[7 8 9]]

[1 2 3]

6

[[1 2]]

[[2 3]

[5 6]]

[[ 1 15 15]

[ 4 15 15]

[ 7 8 9]]

[[0 0 0]

[1 1 1]

[2 2 2]]

[[0 0 0]

[1 1 1]]

[[1 1 1]

[0 0 0]]

**Different functions of Numpy Array**

1 arange 5 random

2 addition 6 maximum

3 sqrt 7 reference for others

4 exp

CODE

**import** numpy **as** np  
A=np.arange(15)  
**print** A  
A=np.arange(1,15,2)  
**print** A  
B=np.sqrt(A)  
**print "B is "  
print** B  
C=np.exp(A)  
**print "C is"  
print** C  
D=np.add(A,B)  
**print "D is"  
print** D  
E=np.maximum(A,B)  
**print "E is"  
print** E

OUTPUT

[ 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14]

[ 1 3 5 7 9 11 13]

B is

[1. 1.73205081 2.23606798 2.64575131 3. 3.31662479

3.60555128]

C is

[2.71828183e+00 2.00855369e+01 1.48413159e+02 1.09663316e+03

8.10308393e+03 5.98741417e+04 4.42413392e+05]

D is

[ 2. 4.73205081 7.23606798 9.64575131 12. 14.31662479

16.60555128]

E is

[ 1. 3. 5. 7. 9. 11. 13.]

**Saving and loading arrays**

Here we made a file named saved\_array which will be saved by extension of .npy

Which is the extension of numpy array. This file will be in getting us the array

CODE

**import** numpy **as** np  
arr=np.arange(10)  
**print** arr  
np.save(**'saved\_array'**,arr)  
new\_array=np.load(**'saved\_array.npy'**)  
**print** new\_array  
array1=np.arange(25)  
array2=np.arange(30)  
  
np.savez(**'saved\_archive.npz'**,x=array1,y=array2)  
load\_archive=np.load(**'saved\_archive.npz'**)  
**print 'load\_archive[x] is'  
print** load\_archive[**'x'**]  
**print 'load\_archive[y] is'  
print** load\_archive[**'y'**]  
np.savetxt(**'notepadfile.txt'**,array1,delimiter=**','**)  
load\_txt\_file=np.loadtxt(**'notepadfile.txt'**,delimiter=**','**)  
**print 'load\_txt\_file is'  
print** load\_txt\_file

OUTPUT

[0 1 2 3 4 5 6 7 8 9]

[0 1 2 3 4 5 6 7 8 9]

load\_archive[x] is

[ 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]

load\_archive[y] is

[ 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]

load\_txt\_file is

[ 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.]

**Statistical and mathematical processing of arrays**

Here matplotlib is used for plotting graphs from the data. Here function is an array which is of size 20x20,since dx and dy have 20 rows and 20 columns respectively. Myfig.png is the file name which gets created and saves the graph in it.

CODE

**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
axes\_values=np.arange(-100,100,10)  
dx,dy=np.meshgrid(axes\_values,axes\_values)  
**print "dx:"  
print** dx  
**print "dy :"  
print** dy  
function=2\*dx+3\*dy  
function2=np.cos(dx)+np.cos(dy)  
  
  
**print** function  
plt.imshow(function2)  
plt.title(**"function cos plot"**)  
plt.colorbar()  
plt.savefig(**'myfig2.png'**)

OUTPUT

dx:

[[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]]

dy :

[[-100 -100 -100 -100 -100 -100 -100 -100 -100 -100 -100 -100 -100 -100

-100 -100 -100 -100 -100 -100]

[ -90 -90 -90 -90 -90 -90 -90 -90 -90 -90 -90 -90 -90 -90

-90 -90 -90 -90 -90 -90]

[ -80 -80 -80 -80 -80 -80 -80 -80 -80 -80 -80 -80 -80 -80

-80 -80 -80 -80 -80 -80]

[ -70 -70 -70 -70 -70 -70 -70 -70 -70 -70 -70 -70 -70 -70

-70 -70 -70 -70 -70 -70]

[ -60 -60 -60 -60 -60 -60 -60 -60 -60 -60 -60 -60 -60 -60

-60 -60 -60 -60 -60 -60]

[ -50 -50 -50 -50 -50 -50 -50 -50 -50 -50 -50 -50 -50 -50

-50 -50 -50 -50 -50 -50]

[ -40 -40 -40 -40 -40 -40 -40 -40 -40 -40 -40 -40 -40 -40

-40 -40 -40 -40 -40 -40]

[ -30 -30 -30 -30 -30 -30 -30 -30 -30 -30 -30 -30 -30 -30

-30 -30 -30 -30 -30 -30]

[ -20 -20 -20 -20 -20 -20 -20 -20 -20 -20 -20 -20 -20 -20

-20 -20 -20 -20 -20 -20]

[ -10 -10 -10 -10 -10 -10 -10 -10 -10 -10 -10 -10 -10 -10

-10 -10 -10 -10 -10 -10]

[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0]

[ 10 10 10 10 10 10 10 10 10 10 10 10 10 10

10 10 10 10 10 10]

[ 20 20 20 20 20 20 20 20 20 20 20 20 20 20

20 20 20 20 20 20]

[ 30 30 30 30 30 30 30 30 30 30 30 30 30 30

30 30 30 30 30 30]

[ 40 40 40 40 40 40 40 40 40 40 40 40 40 40

40 40 40 40 40 40]

[ 50 50 50 50 50 50 50 50 50 50 50 50 50 50

50 50 50 50 50 50]

[ 60 60 60 60 60 60 60 60 60 60 60 60 60 60

60 60 60 60 60 60]

[ 70 70 70 70 70 70 70 70 70 70 70 70 70 70

70 70 70 70 70 70]

[ 80 80 80 80 80 80 80 80 80 80 80 80 80 80

80 80 80 80 80 80]

[ 90 90 90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90]]

[[-500 -480 -460 -440 -420 -400 -380 -360 -340 -320 -300 -280 -260 -240

-220 -200 -180 -160 -140 -120]

[-470 -450 -430 -410 -390 -370 -350 -330 -310 -290 -270 -250 -230 -210

-190 -170 -150 -130 -110 -90]

[-440 -420 -400 -380 -360 -340 -320 -300 -280 -260 -240 -220 -200 -180

-160 -140 -120 -100 -80 -60]

[-410 -390 -370 -350 -330 -310 -290 -270 -250 -230 -210 -190 -170 -150

-130 -110 -90 -70 -50 -30]

[-380 -360 -340 -320 -300 -280 -260 -240 -220 -200 -180 -160 -140 -120

-100 -80 -60 -40 -20 0]

[-350 -330 -310 -290 -270 -250 -230 -210 -190 -170 -150 -130 -110 -90

-70 -50 -30 -10 10 30]

[-320 -300 -280 -260 -240 -220 -200 -180 -160 -140 -120 -100 -80 -60

-40 -20 0 20 40 60]

[-290 -270 -250 -230 -210 -190 -170 -150 -130 -110 -90 -70 -50 -30

-10 10 30 50 70 90]

[-260 -240 -220 -200 -180 -160 -140 -120 -100 -80 -60 -40 -20 0

20 40 60 80 100 120]

[-230 -210 -190 -170 -150 -130 -110 -90 -70 -50 -30 -10 10 30

50 70 90 110 130 150]

[-200 -180 -160 -140 -120 -100 -80 -60 -40 -20 0 20 40 60

80 100 120 140 160 180]

[-170 -150 -130 -110 -90 -70 -50 -30 -10 10 30 50 70 90

110 130 150 170 190 210]

[-140 -120 -100 -80 -60 -40 -20 0 20 40 60 80 100 120

140 160 180 200 220 240]

[-110 -90 -70 -50 -30 -10 10 30 50 70 90 110 130 150

170 190 210 230 250 270]

[ -80 -60 -40 -20 0 20 40 60 80 100 120 140 160 180

200 220 240 260 280 300]

[ -50 -30 -10 10 30 50 70 90 110 130 150 170 190 210

230 250 270 290 310 330]

[ -20 0 20 40 60 80 100 120 140 160 180 200 220 240

260 280 300 320 340 360]

[ 10 30 50 70 90 110 130 150 170 190 210 230 250 270

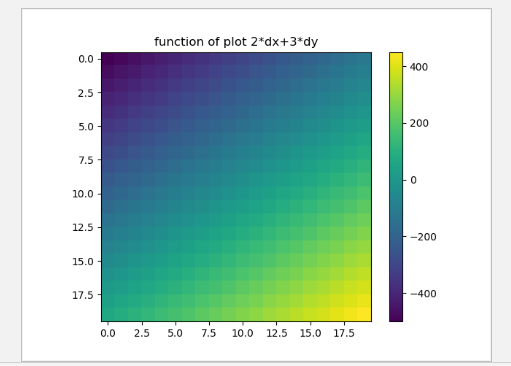
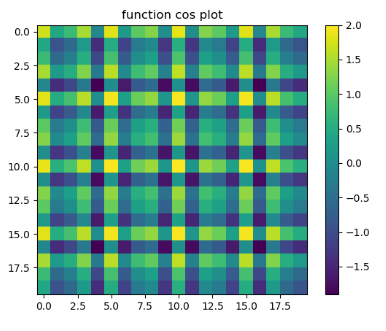
290 310 330 350 370 390]

[ 40 60 80 100 120 140 160 180 200 220 240 260 280 300

320 340 360 380 400 420]

[ 70 90 110 130 150 170 190 210 230 250 270 290 310 330

350 370 390 410 430 450]]

**Conditional clause and Boolean operations**

CODE

**import** numpy **as** np  
x=np.array([100,400,500,600]) *#each no.'a'*y=np.array([10,15,20,25])  
condition =np.array([True,True,False,False])  
z=[a **if** cond **else** b **for** a,cond,b **in** zip(x,condition,y)]  
**print** z  
z2=np.where(condition,x,y)  
**print** z2  
z3=np.where(x>0,0,1)  
**print** z3  
  
**print** x.sum()  
**print** x.sum(0)  
n=np.array([[1,2],[3,4]])  
**print** n.sum(0)  
**print** x.mean()  
**print** x.std()  
**print** x.var()  
condition2=np.array([True,False,True])  
**print** condition2.any()  
**print** condition2.all()  
unsorted\_array=np.array([1,2,8,10,7,3])  
unsorted\_array.sort()  
**print** unsorted\_array  
arr2=np.array([**'solid'**,**'solid'**,**'solid'**,**'liquid'**,**'liquid'**,**'gas'**,**'gas'**])  
**print** np.unique(arr2)  
**print** np.in1d([**'solid'**,**'gas'**,**'plasma'**],arr2)

OUTPUT

[100, 400, 20, 25]

[100 400 20 25]

[0 0 0 0]

1600

1600

[4 6]

400.0

187.08286933869707

35000.0

True

False

[ 1 2 3 7 8 10]

['gas' 'liquid' 'solid']

[ True True False]

**PANDAS**

It is a library which is used to analyse data and perform mathematical operation on data sets

CODE

**import** pandas **as** pd  
**from** pandas **import** Series  
**import** numpy **as** np  
object=Series([5,10,15,20])  
**print** object  
**print** object.values  
**print** object.index  
data\_array=np.array([**'a'**,**'b'**,**'c'**])  
s= Series(data\_array)  
**print** s  
  
s=Series(data\_array,index=[100,101,102])  
**print** s  
s=Series(data\_array,index=[**'index1'**,**'index2'**,**'index3'**])  
**print** s  
revenue=Series([20,80,40,35],index=[**'ola'**,**'uber'**,**'grab'**,**'gojek'**])  
**print** revenue  
  
**print** revenue[**'ola'**]  
**print** revenue[revenue>=35]  
**print 'ola'in** revenue  
**print 'lyft'in** revenue  
revenue\_dict=revenue.to\_dict()  
**print** revenue\_dict  
index\_2=[**'ola'**,**'uber'**,**'grab'**,**'gojek'**,**'lyft'**]  
revenue2=Series(revenue,index\_2)  
**print** revenue2  
**print** pd.isnull(revenue2)  
**print** pd.notnull(revenue2)  
**print** revenue+revenue2  
  
revenue2.name=**"company revenue"**revenue2.index.name=**"company name"  
print** revenue2

OUTPUT

0 5

1 10

2 15

3 20

dtype: int64

[ 5 10 15 20]

RangeIndex(start=0, stop=4, step=1)

0 a

1 b

2 c

dtype: object

100 a

101 b

102 c

dtype: object

index1 a

index2 b

index3 c

dtype: object

ola 20

uber 80

grab 40

gojek 35

dtype: int64

20

uber 80

grab 40

gojek 35

dtype: int64

True

False

{'uber': 80L, 'ola': 20L, 'grab': 40L, 'gojek': 35L}

ola 20.0

uber 80.0

grab 40.0

gojek 35.0

lyft NaN

dtype: float64

ola False

uber False

grab False

gojek False

lyft True

dtype: bool

ola True

uber True

grab True

gojek True

lyft False

dtype: bool

gojek 70.0

grab 80.0

lyft NaN

ola 40.0

uber 160.0

dtype: float64

company name

ola 20.0

uber 80.0

grab 40.0

gojek 35.0

lyft NaN

Name: company revenue, dtype: float64

DATAFRAMES

It is the functionality of pandas.it is a matrix with index and column names.

CODE

**import** pandas **as** pd  
**from** pandas **import** Series,DataFrame  
**import** numpy **as** np  
*#example\_Revenue of companies*revenue\_df=pd.read\_clipboard()  
**print** revenue\_df  
**print** revenue\_df.columns  
**print** revenue\_df[**'Rank'**]  
DataFrame(revenue\_df,columns=[**'Rank'**,**'Name'**,**'Industry'**])  
revenue\_df2=DataFrame(revenue\_df,columns=[**'Rank'**,**'Name'** ,**'Industry'**,**'Profit'**])  
**print** revenue\_df2  
**print** revenue\_df.head[2]  
**print** revenue\_df.tail[2]  
**print** revenue\_df.ix[0]  
**print** revenue\_df.ix[5]  
array1=np.array([1,2,3,4,5,6])  
revenue\_df2[**'Profit'**]=array1  
**print** revenue\_df2  
profits=Series([900,1000],index[3,5])  
revenue\_df2[**'Profit'**]=profits  
**print** revenue\_df2  
**del** revenue\_df2[**'Profit'**]  
**print** revenue\_df2  
  
sample=(  
 **'company'**, [**'A'**,**'B'**],  
 **'Profit'** ,[1000,5000]  
)  
**print** sample  
sample\_df=DataFrame(sample)  
**print** sample\_df

Here we get whatever we have copied in our clipboard

**Index objects**

Indexes are very important part of data analytics

CODE

**import** numpy **as** np  
**import** pandas **as** pd  
**from** pandas **import** Series,DataFrame  
series1=Series([10,20,30,40],index =[**'a'**,**'b'**,**'c'**,**'d'**])  
**print** series1  
index1=series1.index  
**print** index1  
**print** index1[2]  
**print** index1[2:]  
**print** index1[-2:]  
**print** index1[:-2]  
**print** index1[2:3]  
**print** index1[2:4]

OUTPUT

a 10

b 20

c 30

d 40

dtype: int64

Index([u'a', u'b', u'c', u'd'], dtype='object')

c

Index([u'c', u'd'], dtype='object')

Index([u'c', u'd'], dtype='object')

Index([u'a', u'b'], dtype='object')

Index([u'c'], dtype='object')

Index([u'c', u'd'], dtype='object')

**Reindexing methods**

Fillvalue assigns value to the new created index. Ffill is forward fill which give values in forward manner.

CODE

**import** numpy **as** np  
**import** pandas **as** pd  
**from** pandas **import** Series,DataFrame  
**from** numpy.random **import** random  
  
series1=Series([1,2,3,4],index=[**'e'**,**'f'**,**'g'**,**'h'**])  
**print** series1  
series2=series1.reindex([**'e'**,**'f'**,**'g'**,**'h'**,**'i'**,**'j'**])  
**print** series2  
series2=series2.reindex([**'e'**,**'f'**,**'g'**,**'h'**,**'i'**,**'j'**,**'k'**],fill\_value=10)  
**print** series2  
cars=Series([**'audi'**,**'merc'**,**'bmw'**],index=[0,4,8])  
**print** cars  
ranger=range(13)  
**print** ranger  
cars= cars.reindex(ranger,method=**'ffill'**)  
**print** cars  
df\_1 =DataFrame(random(25).reshape (5,5),index=[**'a'**,**'b'**,**'c'**,**'d'**,**'e'**],columns=[**'c1'**,**'c2'**,**'c3'**,**'c4'**,**'c5'**])  
**print** df\_1  
df\_2=df\_1.reindex([**'a'**,**'b'**,**'c'**,**'d'**,**'e'**,**'f'**])  
**print** df\_2  
df\_3=df\_2.reindex(columns=[**'c1'**,**'c2'**,**'c3'**,**'c4'**,**'c5'**,**'c6'**])  
**print** df\_3  
df\_4= df\_1.ix[[**'a'**,**'b'**,**'c'**,**'d'**,**'e'**,**'f'**],[**'c1'**,**'c2'**,**'c3'**,**'c4'**,**'c5'**,**'c6'**]]  
**print** df\_4

OUTPUT

e 1

f 2

g 3

h 4

dtype: int64

e 1.0

f 2.0

g 3.0

h 4.0

i NaN

j NaN

dtype: float64

e 1.0

f 2.0

g 3.0

h 4.0

i NaN

j NaN

k 10.0

dtype: float64

0 audi

4 merc

8 bmw

dtype: object

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

0 audi

1 audi

2 audi

3 audi

4 merc

5 merc

6 merc

7 merc

8 bmw

9 bmw

10 bmw

11 bmw

12 bmw

dtype: object

c1 c2 c3 c4 c5

a 0.178133 0.226698 0.029846 0.094328 0.760370

b 0.247093 0.447577 0.226775 0.570526 0.325021

c 0.474253 0.724337 0.534604 0.839422 0.814764

d 0.596021 0.343841 0.275155 0.201213 0.115745

e 0.885977 0.482294 0.343851 0.026970 0.892573

c1 c2 c3 c4 c5

a 0.178133 0.226698 0.029846 0.094328 0.760370

b 0.247093 0.447577 0.226775 0.570526 0.325021

c 0.474253 0.724337 0.534604 0.839422 0.814764

d 0.596021 0.343841 0.275155 0.201213 0.115745

e 0.885977 0.482294 0.343851 0.026970 0.892573

f NaN NaN NaN NaN NaN

c1 c2 c3 c4 c5 c6

a 0.178133 0.226698 0.029846 0.094328 0.760370 NaN

b 0.247093 0.447577 0.226775 0.570526 0.325021 NaN

c 0.474253 0.724337 0.534604 0.839422 0.814764 NaN

d 0.596021 0.343841 0.275155 0.201213 0.115745 NaN

e 0.885977 0.482294 0.343851 0.026970 0.892573 NaN

f NaN NaN NaN NaN NaN NaN

Drooping entries from datatypes

CODE

**import** numpy **as** np  
**import** pandas **as** pd  
**from** pandas **import** Series,DataFrame  
cars=Series([**'bmw'**,**'audi'**,**'merc'**],index=[**'a'**,**'b'**,**'c'**])  
**print** cars  
cars=cars.drop(**'a'**)  
**print** cars  
cars\_df=DataFrame(np.arange(9).reshape (3,3),index=[**'bmw'**,**'audi'**,**'merc'**],columns=[**'rev'**,**'profit'**,**'exp'**])  
**print** cars\_df  
  
cars\_df=cars\_df.drop(**'bmw'**,axis=0)  
**print** cars\_df  
cars\_df=cars\_df.drop(**'profit'**,axis=1)  
**print** cars\_df

OUTPUT

a bmw

b audi

c merc

dtype: object

b audi

c merc

dtype: object

rev profit exp

bmw 0 1 2

audi 3 4 5

merc 6 7 8

rev profit exp

audi 3 4 5

merc 6 7 8

rev exp

audi 3 5

merc 6 8

**Handling null data in pandas**

import numpy as np  
import pandas as pd  
from pandas import Series,DataFrame  
series1=Series(['A','B','C','D',np.nan])  
print series1.isnull()  
print series1.dropna()  
df\_1=DataFrame([[1,2,3],[5,6,np.nan],[7,np.nan,10],[np.nan,np.nan,np.nan]])  
print df\_1  
print df\_1.dropna()  
print df\_1.dropna(how='all')  
print df\_1.dropna(axis=1)  
df2=DataFrame([[1,2,3,np.nan],[4,5,6,7],[8,9,np.nan,np.nan],[12,np.nan,np.nan,np.nan]])  
print df2  
print df2.dropna(thresh=3)  
print df2.dropna(thresh=2)  
  
print df2.fillna(0)  
print df2.fillna({0:0,1:50,2:100,3:200})

OUTPUT

0 False

1 False

2 False

3 False

4 True

dtype: bool

0 A

1 B

2 C

3 D

dtype: object

0 1 2

0 1.0 2.0 3.0

1 5.0 6.0 NaN

2 7.0 NaN 10.0

3 NaN NaN NaN

0 1 2

0 1.0 2.0 3.0

0 1 2

0 1.0 2.0 3.0

1 5.0 6.0 NaN

2 7.0 NaN 10.0

Empty DataFrame

Columns: []

Index: [0, 1, 2, 3]

0 1 2 3

0 1 2.0 3.0 NaN

1 4 5.0 6.0 7.0

2 8 9.0 NaN NaN

3 12 NaN NaN NaN

0 1 2 3

0 1 2.0 3.0 NaN

1 4 5.0 6.0 7.0

0 1 2 3

0 1 2.0 3.0 NaN

1 4 5.0 6.0 7.0

2 8 9.0 NaN NaN

0 1 2 3

0 1 2.0 3.0 0.0

1 4 5.0 6.0 7.0

2 8 9.0 0.0 0.0

3 12 0.0 0.0 0.0

0 1 2 3

0 1 2.0 3.0 200.0

1 4 5.0 6.0 7.0

2 8 9.0 100.0 200.0

3 12 50.0 100.0 200.0

**Selecting and modifying data in pandas**

CODE

import numpy as np  
import pandas as pd  
from pandas import Series,DataFrame  
series1=Series([100,200,300],index=['A','B','C'])  
print series1['A']  
print series1['B']  
print series1[['A','B']]  
print series1[0]  
print series1[0:2]  
  
print series1[series1>150]  
print series1[series1==300]  
df1=DataFrame(np.arange(9).reshape(3,3),index=['car','cycle','bike'],columns=['A','B','C'])  
print df1['A']  
print df1[['A','B']]  
print df1>5  
  
print df1.ix['cycle']  
print df1.ix[1]

CODE

100

200

A 100

B 200

dtype: int64

100

A 100

B 200

dtype: int64

B 200

C 300

dtype: int64

C 300

dtype: int64

car 0

cycle 3

bike 6

Name: A, dtype: int32

A B

car 0 1

cycle 3 4

bike 6 7

A B C

car False False False

cycle False False False

bike True True True

A 3

B 4

C 5

Name: cycle, dtype: int32

A 3

B 4

C 5

Name: cycle, dtype: int32

**Co-ordinate and regulate data**

CODE

**import** numpy **as** np  
**import** pandas **as** pd  
**from** pandas **import** Series,DataFrame  
sera=Series([100,200,300],index=[**'a'**,**'b'**,**'c'**])  
serb=Series([300,400,500,600],index=[**'a'**,**'b'**,**'c'**,**'d'**])  
**print** sera+serb  
df1=DataFrame(np.arange(4).reshape(2,2),columns=[**'a'**,**'b'**],index=[**'car'**,**'bike'**])  
df2=DataFrame(np.arange(9).reshape(3,3),columns=[**'a'**,**'b'**,**'c'**],index=[**'car'**,**'bike'**,**'cycle'**])  
**print** df1  
**print** df2  
**print** df1+df2  
df1=df1.add(df2,fill\_value=0)  
**print** df1  
serc=df2.ix[0]  
**print** df2-serc

OUTPUT

a 400.0

b 600.0

c 800.0

d NaN

dtype: float64

a b

car 0 1

bike 2 3

a b c

car 0 1 2

bike 3 4 5

cycle 6 7 8

a b c

bike 5.0 7.0 NaN

car 0.0 2.0 NaN

cycle NaN NaN NaN

a b c

bike 5.0 7.0 5.0

car 0.0 2.0 2.0

cycle 6.0 7.0 8.0

a b c

car 0 0 0

bike 3 3 3

cycle 6 6 6

**statistics and graph sketches with pandas**

CODE

**import** numpy **as** np  
**import** pandas **as** pd  
**from** numpy.random **import** randn  
**from** pandas **import** Series,DataFrame  
**import** matplotlib.pyplot **as** plt  
array1=np.array([[10,np.nan,20],[30,40,np.nan]])  
**print** array1  
df1=DataFrame(array1,index=[1,2],columns=list(**'ABC'**))  
**print** df1  
  
**print** df1.sum()  
**print** df1.sum(axis=1)  
**print** df1.min()  
**print** df1.max()  
**print** df1.idxmax()  
**print** df1.cumsum()  
**print** df1.describe()  
df2=DataFrame(randn(9).reshape(3,3),index=[1,2,3],columns=list(**'ABC'**))  
**print** df2  
  
plt.plot(df2)  
plt.legend(df2.columns,loc=**"lower right"**)  
plt.savefig(**'samplogic.png'**)  
plt.show()  
series1=Series(list(**'abccaabd'**))  
**print** series1.unique()  
**print** series1.value\_counts()

OUTPUT

[[10. nan 20.]

[30. 40. nan]]

A B C

1 10.0 NaN 20.0

2 30.0 40.0 NaN

A 40.0

B 40.0

C 20.0

dtype: float64

1 30.0

2 70.0

dtype: float64

A 10.0

B 40.0

C 20.0

dtype: float64

A 30.0

B 40.0

C 20.0

dtype: float64

A 2

B 2

C 1

dtype: int64

A B C

1 10.0 NaN 20.0

2 40.0 40.0 NaN

A B C

count 2.000000 1.0 1.0

mean 20.000000 40.0 20.0

std 14.142136 NaN NaN

min 10.000000 40.0 20.0

25% 15.000000 40.0 20.0

50% 20.000000 40.0 20.0

75% 25.000000 40.0 20.0

max 30.000000 40.0 20.0

A B C

1 -0.539108 0.280789 -1.819165

2 -0.400573 -2.154266 -0.165173

3 -0.801443 -0.123627 1.922660

