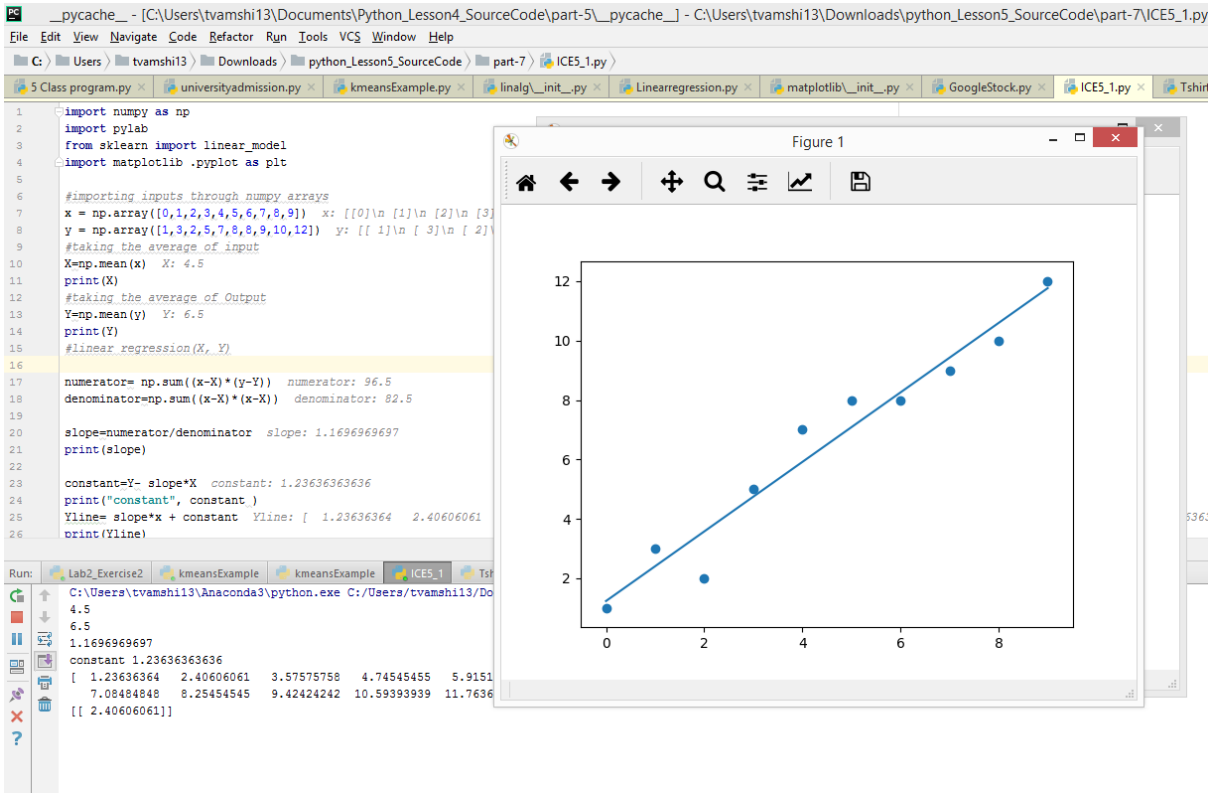
```python
import numpy as np
import pylab
from sklearn import linear_model
import matplotlib .pyplot as plt

#importing inputs through numpy arrays
x = np.array([0,1,2,3,4,5,6,7,8,9])
y = np.array([1,3,2,5,7,8,8,9,10,12])
#taking the average of input
X=np.mean(x)
print(X)
#taking the average of Output
Y=np.mean(y)
print(Y)
#linear regression(X, Y)

numerator= np.sum((x-X)*(y-Y))
denominator=np.sum((x-X)*(x-X))

slope=numerator/denominator
print(slope)

constant=Y- slope*X
print("constant", constant )
Yline= slope*x + constant
print(Yline)

linear_mod = linear_model.LinearRegression()
x = np.reshape(x,(len(x),1))
y = np.reshape(y,(len(y),1))
linear_mod.fit(x,y)
predicted_output = linear_mod.predict(1)
print(predicted_output)
plt.plot(x,Yline)
plt.scatter(x,y)
plt.show()
```



```
import numpy as np  # import modules
import matplotlib.pyplot as plt # matplotlib
import random            # random

def centroid_content(X, mu):    # centroid function
    centroid = {}           # initiation
    for x in X:         # forloop to enumerate the  the array
        value = min([(i[0],np.linalg.norm(x - mu[i[0]]))for i in enumerate(mu)],
key=lambda s:s[1])[0]
        try:                    # try block for exception
            centroid[value].append(x)
        except:
            centroid[value] = [x]
    return centroid


def ne_center(mu, centroid):      # function for new centroid
    keya =sorted(centroid.keys())     # sort for keys
    nemu = np.array([(np.mean(centroid[k],axis = 0))for k in keya])
    return nemu

def match(nemu, olmu):            # fucntion for comparison
    return (set([tuple(a)for a in nemu]) == set([tuple(a)for a in olmu]))

def Kmeans(X, K, N):   # definition of fucntion
    temp1 = np.random.randint(N, size = K)    # creation of temp1
    olmu = np.array([X[i]for i in temp1])
    temp2 = np.random.randint(N, size=K)    # creation of temp2
    nemu = np.array([X[i] for i in temp2])
    centroid = centroid_content(X, olmu)
    itr = 0                     # initializing iteration
    plot_cluster(olmu,centroid,itr)
    while not match(nemu, olmu):
        itr = itr + 1
        olmu = nemu
        cluster = centroid_content(X,nemu)
        plot_cluster(nemu, cluster,itr)
        nemu = ne_center(nemu, cluster)
    plot_cluster(nemu, cluster, itr)
    return

def plot_cluster(mu,cluster, itr):  # plotting the cluster using colors
    color = 10 * ['r.','g.','k.','b.','m.']
    print('Iteration number : ',itr)      # print of iteration number
    for l in cluster.keys():
        for m in range(len(cluster[l])):
            plt.plot(cluster[l][m][0], cluster[l][m][1], color[l], markersize=10) #
use of plt.plot from matlib
    plt.scatter(mu[:,0],mu[:,1],marker = '*', s = 150, linewidths = 5, zorder = 10)
# scattering using  line widths
    plt.show()

def in_graph(N, p1, p2):       # defining a graph fucntion
    X = np.array([(random.uniform(p1,p2),random.uniform(p1,p2))for i in range(N)])
# use of np array from numpy
    return X


def My_Clusters(): # fucntion for clusters
# asking the inputs and typecasting them
    N = int(input('Enter the number of datapoints:'))
    K = int(input('Enter the number of Centroids:'))

    p1 = int(input('The lower bound:'))
    p2 = int(input('The upper bound:'))
    X = in_graph(N, p1, p2)
 #plotting the points
    plt.scatter(X[:, 0], X[:, 1])
```

```
    plt.show()
    temp = Kmeans(X, K, N)


if __name__ == '__main__':          # main function
    My_Clusters()
```