PPiC 7.4 A palindrome is a word that is spelled the same forward and backward, but I also take punctuations into account. I will write a function using *while* loop so that if a word is palindrome, the function return `True`, if it is not a palindrome, the function will return `False`.

```
>>> import string #import string module in Python
>>> def remove_punctuations(word): #define function called remvoe_punctuations to remove the punctuations in parameter word
    return "".join(i.lower() for i in word if i in string.ascii_letters) #turn word to lowercase, only return the string in ascii_letters

>>> def isPalindrome(S): #define function called isPalindrome to check if parameter S is a palindrome.
        S = remove_punctuations(S) #First, use function remove_punctuations to remove the punctuations in S.
        while S == S[::-1]: #while S equals the reverse of S,
                return print("True") #S is a palindrome so return True.
        else: #other cases means they are not equal, then it is not a palindrome, so return False.
                return print("False")


>>> S = "A man, a plan, a canal: Panama." #Test the function.
>>> isPalindrome(S)
True
>>> S = "Hello World!"
>>> isPalindrome(S)
False
>>>
```

PPiC 7.5 Load and run the `clusterAnalysis` function using the exam score data. Compare your clusters with those shown in Session 7.1.

```
>>> import random
>>> import math
>>> def euclidD(point1, point2):
    sum = 0
    # Go through each dimension and sum of the squares of the
    # differences.
    for index in range(len(point1)):
        diff = (point1[index] - point2[index]) ** 2
        sum += diff

    return math.sqrt(sum)

>>> def readFile(filename):
    # open the file for reading
    datafile = open(filename, "r")
    # create an empty data dictionary and initial key
    datadict = {}
    key = 0

    # Go through each line in the file, assuming that each line
    # has one integer on it.
    for aline in datafile:
       key += 1
       score = int(aline)
       # Store that integer in a data dictionary keyed by line number.
       datadict[key] = [score]

    return datadict



>>> def createCentroids(k, datadict):
    # Initially centroids and centroid keys are empty lists.
    centroids=[]
    centroidKeys = []

    # Use a while loop to keep looping until k centroids are found.
    while len(centroids) < k:
       # Randomly choose a key.
       rkey = random.randint(1,len(datadict))
       # If that key isn't already a centroid, make it be one.
       if rkey not in centroidKeys:
           centroids.append(datadict[rkey])
           centroidKeys.append(rkey)

    # Return just the list of centroids
    return centroids
```

```
>>> def createClusters(k, centroids, datadict, repeats):
    # Start a new pass through the data.
    for apass in range(repeats):
        print("****PASS",apass,"****")
        # Start with empty clusters.
        clusters = []
        for i in range(k):
           clusters.append([])

        # Go through each piece of data in the data dictionary and
        # find the centroid that it is closest to using Euclidean
        # distance.  Store the key with the cluster.
        for akey in datadict:
           distances = []
           for clusterIndex in range(k):
               dist = euclidD(datadict[akey],centroids[clusterIndex])
               distances.append(dist)

           mindist = min(distances)
           index = distances.index(mindist)

           clusters[index].append(akey)

        # Now go through each cluster
        dimensions = len(datadict[1])
        for clusterIndex in range(k):
           # initialize a list of dimension length of all zeros.
           sums = [0]*dimensions
           # go through each data point in the cluster and sum
           # up the values for each dimension
           for akey in clusters[clusterIndex]:
               datapoints = datadict[akey]
               for ind in range(len(datapoints)):
                   sums[ind] = sums[ind] + datapoints[ind]
           # Now just find the averages along each dimension and
           # that point becomes the new centroid.
           for ind in range(len(sums)):
               clusterLen = len(clusters[clusterIndex])
               if clusterLen != 0:
                  sums[ind] = sums[ind]/clusterLen

           centroids[clusterIndex] = sums

        # Print out the data in each cluster at the end of each pass.
        for c in clusters:
           print ("CLUSTER")
           for key in c:
               print(datadict[key], end=" ")
           print()

    return clusters
```

```
>>> def clusterAnalysis(dataFile):
        examDict = readFile(dataFile)
        examCentroids = createCentroids(5, examDict)
        examClusters = createClusters(5, examCentroids, examDict, 3)

>>> clusterAnalysis("cs150exams.txt")
****PASS 0 ****
CLUSTER
[34] [44] [45] [24] [34] [45]
CLUSTER
[87] [78] [78]
CLUSTER
[12] [12] [22] [11]
CLUSTER
[56] [76] [76] [77]
CLUSTER
[98] [98] [90] [89]
****PASS 1 ****
CLUSTER
[34] [44] [45] [34] [45]
CLUSTER
[87] [78] [78] [77]
CLUSTER
[12] [24] [12] [22] [11]
CLUSTER
[56] [76] [76]
CLUSTER
[98] [98] [90] [89]
****PASS 2 ****
CLUSTER
[34] [44] [45] [34] [45]
CLUSTER
[76] [76] [78] [78] [77]
CLUSTER
[12] [24] [12] [22] [11]
CLUSTER
[56]
CLUSTER
[87] [98] [98] [90] [89]
>>> #Compare to the clusters of exam scores in Session 7.1, we have less cluster with just one score.
#most of them have about three to five in the cluster.
```

PPiC 7.6 Run `clusterAnalysis` again but try to use different numbers of clusters and passes.

```
>>> def clusterAnalysis(dataFile): #Changing cluster to 4.
        examDict = readFile(dataFile)
        examCentroids = createCentroids(4, examDict)
        examClusters = createClusters(4, examCentroids, examDict, 3)


>>> clusterAnalysis("cs150exams.txt")
****PASS 0 ****
CLUSTER
[12] [12] [22] [11]
CLUSTER
[56] [87] [76] [98] [76] [78] [98] [78] [90] [89] [77]
CLUSTER
[34] [44] [45] [24] [34] [45]
CLUSTER

****PASS 1 ****
CLUSTER
[12] [24] [12] [22] [11]
CLUSTER
[87] [76] [98] [76] [78] [98] [78] [90] [89] [77]
CLUSTER
[34] [56] [44] [45] [34] [45]
CLUSTER

****PASS 2 ****
CLUSTER
[12] [24] [12] [22] [11]
CLUSTER
[87] [76] [98] [76] [78] [98] [78] [90] [89] [77]
CLUSTER
[34] [56] [44] [45] [34] [45]
CLUSTER
```

```
>>> def clusterAnalysis(dataFile): #Changing passes to 2.
        examDict = readFile(dataFile)
        examCentroids = createCentroids(4, examDict)
        examClusters = createClusters(4, examCentroids, examDict, 2)


>>> clusterAnalysis("cs150exams.txt")
****PASS 0 ****
CLUSTER
[12] [12] [22] [11]
CLUSTER
[56] [44] [45] [45]
CLUSTER
[87] [76] [98] [76] [78] [98] [78] [90] [89] [77]
CLUSTER
[34] [24] [34]
****PASS 1 ****
CLUSTER
[12] [12] [22] [11]
CLUSTER
[56] [44] [45] [45]
CLUSTER
[87] [76] [98] [76] [78] [98] [78] [90] [89] [77]
CLUSTER
[34] [24] [34]
```

PS. 5.6.8 Consider the list of characters: `['P','Y','T','H','O','N']`. Show how this list is sorted using the following algorithms:

```
>>> #Bubble Sort
>>> def bubbleSort(alist):
    for passnum in range(len(alist)-1,0,-1):
        for i in range(passnum):
            if alist[i]>alist[i+1]:
                temp = alist[i]
                alist[i] = alist[i+1]
                alist[i+1] = temp


>>> alist = ['P','Y','T','H','O','N']
>>> bubbleSort(alist)
>>> print(alist)
['H', 'N', 'O', 'P', 'T', 'Y']
>>>
>>> #Selection Sort
['H', 'N', 'O', 'P', 'T', 'Y']
>>> def selectionSort(alist):
   for fillslot in range(len(alist)-1,0,-1):
       positionOfMax=0
       for location in range(1,fillslot+1):
           if alist[location]>alist[positionOfMax]:
               positionOfMax = location

       temp = alist[fillslot]
       alist[fillslot] = alist[positionOfMax]
       alist[positionOfMax] = temp

>>> alist = ['P','Y','T','H','O','N']
>>> selectionSort(alist)
>>> print(alist)
['H', 'N', 'O', 'P', 'T', 'Y']
```