6.00 Handout, Lecture 20
(Not intended to make sense outside of lecture)

```
class Point(object):

class Cluster(object):

class ClusterSet(object):

class Mammal(Point):
    def __init__(self, name, originalAttrs, scaledAttrs = None):
        Point.__init__(self, name, originalAttrs, originalAttrs)
    def scaleFeatures(self, key):
        scaleDict = {'identity': [1,1,1,1,1,1,1,1],
                     '1/max': [1/3.0,1/4.0,1.0,1.0,1/4.0,1/4.0,1/6.0,1/6.0]}
        scaledFeatures = []
        features = self.getOriginalAttrs()
        for i in range(len(features)):
            scaledFeatures.append(features[i]*scaleDict[key][i])
        self.attrs = scaledFeatures

def buildMammalPoints(fName, scaling):
    nameList, featureList = readMammalData(fName)
    points = []
    for i in range(len(nameList)):
        point = Mammal(nameList[i], pylab.array(featureList[i]))
        point.scaleFeatures(scaling)
        points.append(point)
    return points

#Use hierarchical clustering for mammals teeth
def test0(numClusters = 2, scaling = 'identity', printSteps = False,
          printHistory = True):
    points = buildMammalPoints('mammalTeeth.txt', scaling)
    cS = ClusterSet(Mammal)
    for p in points:
        cS.add(Cluster([p], Mammal))
    history = cS.mergeN(Cluster.maxLinkageDist, numClusters,
                        toPrint = printSteps)
    if printHistory:
        …
    clusters = cS.getClusters()
    print 'Final set of clusters:'
    index = 0
    for c in clusters:
        print '  C' + str(index) + ':', c
        index += 1
```

```
def kmeans(points, k, cutoff, pointType, maxIters = 100,
           toPrint = False):
    #Get k randomly chosen initial centroids
    initialCentroids = random.sample(points, k)
    clusters = []
    #Create a singleton cluster for each centroid
    for p in initialCentroids:
        clusters.append(Cluster([p], pointType))
    numIters = 0
    biggestChange = cutoff
    while biggestChange >= cutoff and numIters < maxIters:
        #Create a list containing k empty lists
        newClusters = []
        for i in range(k):
            newClusters.append([])
        for p in points:
            #Find the centroid closest to p
            smallestDistance = p.distance(clusters[0].getCentroid())
            index = 0
            for i in range(k):
                distance = p.distance(clusters[i].getCentroid())
                if distance < smallestDistance:
                    smallestDistance = distance
                    index = i
            #Add p to the list of points for the appropriate cluster
            newClusters[index].append(p)
        #Update each cluster and record how much the centroid has changed
        biggestChange = 0.0
        for i in range(len(clusters)):
            change = clusters[i].update(newClusters[i])
            biggestChange = max(biggestChange, change)
        numIters += 1
    #Calculate the coherence of the least coherent cluster
    maxDist = 0.0
    for c in clusters:
        for p in c.members():
            if p.distance(c.getCentroid()) > maxDist:
                maxDist = p.distance(c.getCentroid())
    print 'Number of iterations =', numIters, 'Max Diameter =', maxDist
    return clusters, maxDist
```