6.00 Handout, Lecture 21
(Not intended to make sense outside of lecture)

```
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())
    return clusters, maxDist
```

```python
class Node(object):
    def __init__(self, name):
        self.name = name
    def getName(self):
        return self.name
    def __str__(self):
        return self.name

class Edge(object):
    def __init__(self, src, dest, weight = 0):
        self.src = src
        self.dest = dest
        self.weight = weight
    def getSource(self):
        return self.src
    def getDestination(self):
        return self.dest
    def getWeight(self):
        return self.weight
    def __str__(self):
        return str(self.src) + '->' + str(self.dest)

class Digraph(object):
    def __init__(self):
        self.nodes = set([])
        self.edges = {}
    def addNode(self, node):
        if node.getName() in self.nodes:
            raise ValueError('Duplicate node')
        else:
            self.nodes.add(node)
            self.edges[node] = []
    def addEdge(self, edge):
        src = edge.getSource()
        dest = edge.getDestination()
        if not(src in self.nodes and dest in self.nodes):
            raise ValueError('Node not in graph')
        self.edges[src].append(dest)
    def childrenOf(self, node):
        return self.edges[node]
    def hasNode(self, node):
        return node in self.nodes
    def __str__(self):
        res = ''
        for k in self.edges:
            for d in self.edges[k]:
                res = res + str(k) + '->' + str(d) + '\n'
        return res[:-1]

class Graph(Digraph):
    def addEdge(self, edge):
        Digraph.addEdge(self, edge)
        rev = Edge(edge.getDestination(), edge.getSource())
        Digraph.addEdge(self, rev)
```