```
from math import pi, sin, cos
from collections import namedtuple
from random import random, choice
from copy import copy

try:
    import psyco
    psyco.full()
except ImportError:
    pass


FLOAT_MAX = 1e100


class Point:
    __slots__ = ["x", "y", "group"]
    def __init__(self, x=0.0, y=0.0, group=0):
        self.x, self.y, self.group = x, y, group


def generate_points(npoints, radius):
    points = [Point() for _ in xrange(npoints)]

    # note: this is not a uniform 2-d distribution
    for p in points:
        r = random() * radius
        ang = random() * 2 * pi
        p.x = r * cos(ang)
        p.y = r * sin(ang)

    return points


def nearest_cluster_center(point, cluster_centers):
    """Distance and index of the closest cluster center"""
    def sqr_distance_2D(a, b):
        return (a.x - b.x) ** 2  +  (a.y - b.y) ** 2

    min_index = point.group
    min_dist = FLOAT_MAX

    for i, cc in enumerate(cluster_centers):
        d = sqr_distance_2D(cc, point)
        if min_dist > d:
            min_dist = d
            min_index = i

return (min_index, min_dist)


def kpp(points, cluster centers):
```

```
    cluster_centers[0] = copy(choice(points))
    d = [0.0 for _ in xrange(len(points))]

    for i in xrange(1, len(cluster_centers)):
        sum = 0
        for j, p in enumerate(points):
            d[j] = nearest_cluster_center(p, cluster_centers[:i])[1]
            sum += d[j]

        sum *= random()

        for j, di in enumerate(d):
            sum -= di
            if sum > 0:
                continue
            cluster_centers[i] = copy(points[j])
            break

for p in points:
    p.group = nearest_cluster_center(p, cluster_centers)[0]


def lloyd(points, nclusters):
    cluster_centers = [Point() for _ in xrange(nclusters)]

    # call k++ init
    kpp(points, cluster_centers)

    lenpts10 = len(points) >> 10

    changed = 0
    while True:
        # group element for centroids are used as counters
        for cc in cluster_centers:
            cc.x = 0
            cc.y = 0
            cc.group = 0

        for p in points:
            cluster_centers[p.group].group += 1
            cluster_centers[p.group].x += p.x
            cluster_centers[p.group].y += p.y

        for cc in cluster_centers:
            cc.x /= cc.group
            cc.y /= cc.group

        # find closest centroid of each PointPtr
        changed = 0
        for p in points:
            min_i = nearest_cluster_center(p, cluster_centers)[0]
            if min_i != p.group:
                changed += 1
                p.group = min_i

        # stop when 99.9% of points are good
```

```
        if changed <= lenpts10:
            break

for i, cc in enumerate(cluster_centers):
    cc.group = i

    return cluster_centers


def print_eps(points, cluster_centers, W=400, H=400):
    Color = namedtuple("Color", "r g b");

    colors = []
    for i in xrange(len(cluster_centers)):
        colors.append(Color((3 * (i + 1) % 11) / 11.0,
                            (7 * i % 11) / 11.0,
                            (9 * i % 11) / 11.0))

    max_x = max_y = -FLOAT_MAX
    min_x = min_y = FLOAT_MAX

for p in points:
    if max_x < p.x: max_x = p.x
        if min_x > p.x: min_x = p.x
        if max_y < p.y: max_y = p.y
        if min_y > p.y: min_y = p.y

    scale = min(W / (max_x - min_x),
                H / (max_y - min_y))
        cx = (max_x + min_x) / 2
cy = (max_y + min_y) / 2

    print "%%!PS-Adobe-3.0\n%%%%BoundingBox: -5 -5 %d %d" % (W + 10, H + 10)

    print ("/l {rlineto} def /m {rmoveto} def\n" +
           "/c { .25 sub exch .25 sub exch .5 0 360 arc fill } def\n" +
           "/s { moveto -2 0 m 2 2 l 2 -2 l -2 -2 l closepath " +
           "   gsave 1 setgray fill grestore gsave 3 setlinewidth" +
           " 1 setgray stroke grestore 0 setgray stroke }def")

        for i, cc in enumerate(cluster_centers):
            print ("%g %g %g setrgbcolor" %
                   (colors[i].r, colors[i].g, colors[i].b))

                for p in points:
                    if p.group != i:
                        continue
                            print ("%.3f %.3f c" % ((p.x - cx) * scale + W / 2,
                                                    (p.y - cy) * scale + H / 2))

                                print ("\n0 setgray %g %g s" % ((cc.x - cx) *
scale + W / 2,
                                                               (cc.y - cy) *
scale + H / 2))

print "\n%%%%EOF"
```

```
def main():
    npoints = 30000
    k = 7 # # clusters

    points = generate_points(npoints, 10)
    cluster_centers = lloyd(points, k)
    print_eps(points, cluster_centers)


main()
```