CG Assignment 1

Divyang Mittal (17CS10012)
Amatya Sharma (17CS30042)

Problem Statement :

- Generate instances of n random points
- $(n = 10r + 3, r > 1)$ on the plane.
- Construct a separating circle that encloses 30% of points in the interior
- Vary r, report your results and show diagrams
- Submit report via Moodle
- Due: 19 January 2021
- Credit: 5%

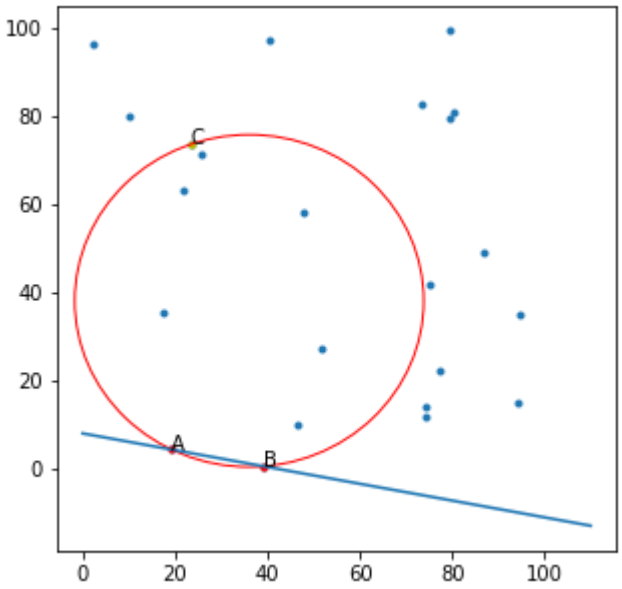
Assumptions :

- Dimensions of the defining plane $N \times N = 100 \times 100$.
- No two points have same x (or y) coordinates.
- No 4 points are concyclic
- No 3 points are colinear

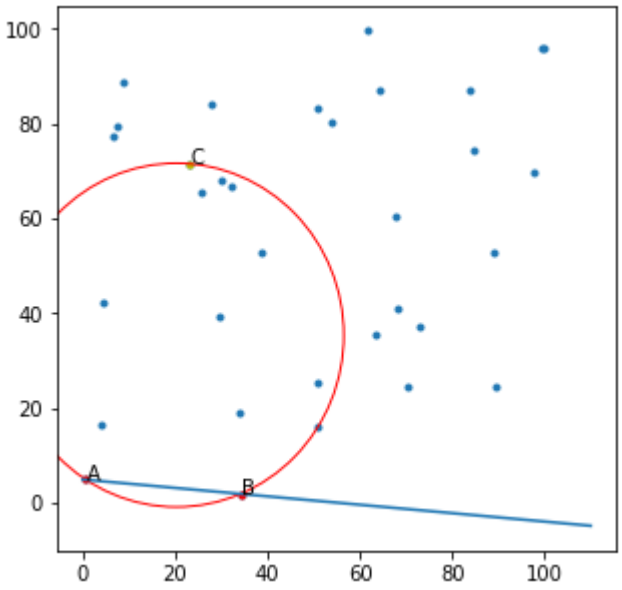
Outputs ::: Approach 1 ($\frac{3n}{10}^{th}$ largest angle)

In [13]:
```
main_angle(2); main_angle(3); main_angle(4); main_angle(5); main_angle(10); ma
in_angle(50); main_angle(75); main_angle(100); # main(-1)
```

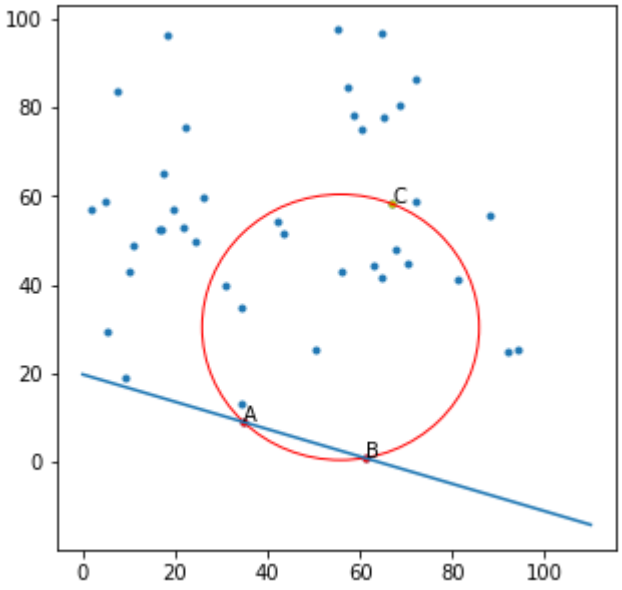
```
---------------
 Separating Circle of  23  points:
```



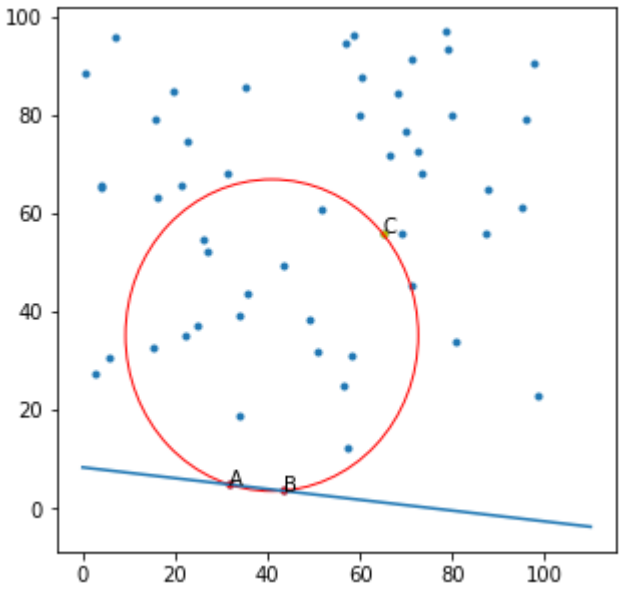

```
Percentage of points inside the circle =  36.36363636363637
---------------
 Separating Circle of  33  points:
```



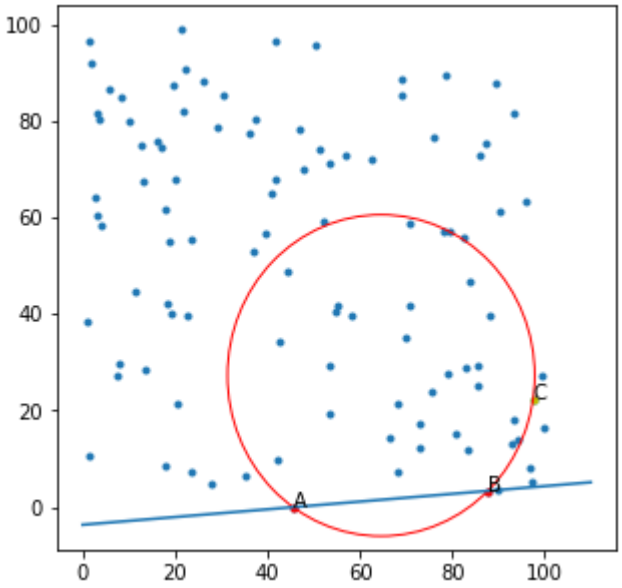

```
Percentage of points inside the circle =  29.03225806451613
---------------
 Separating Circle of  43  points:
```



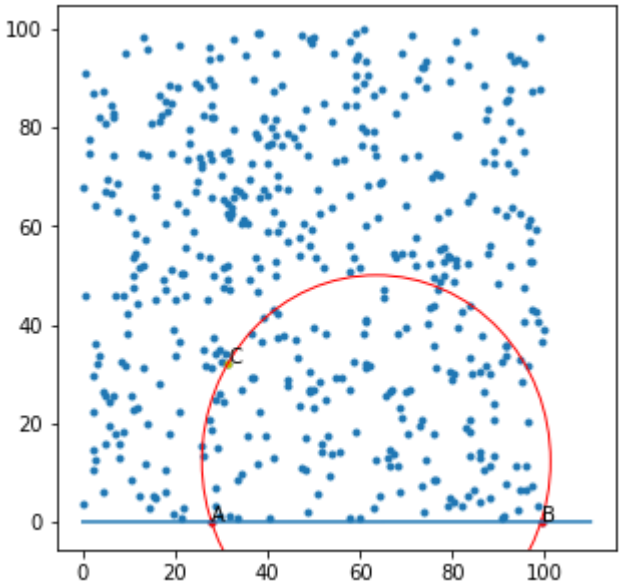

```
Percentage of points inside the circle =  33.33333333333333
---------------
 Separating Circle of  53  points:
```



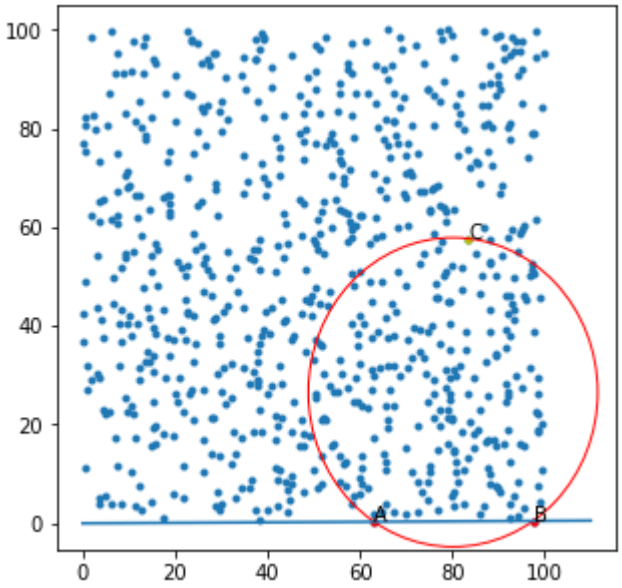

```
Percentage of points inside the circle =  32.69230769230769
---------------
 Separating Circle of  103  points:
```



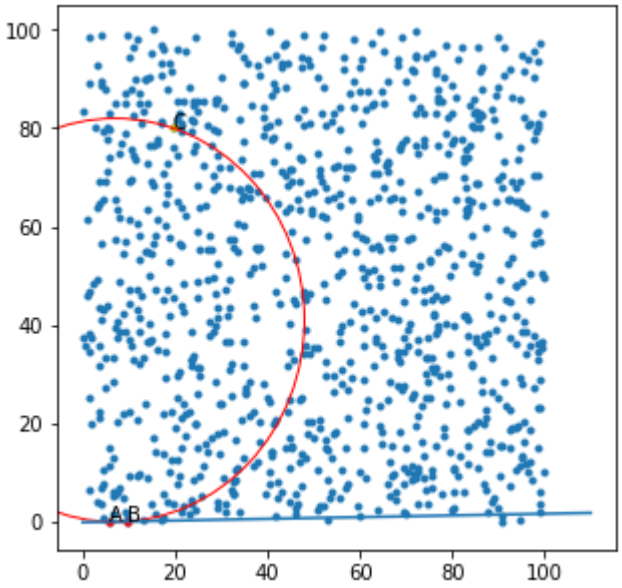

```
Percentage of points inside the circle =  29.411764705882355
---------------
 Separating Circle of  503  points:
```




```
Percentage of points inside the circle =  30.0796812749004
---------------
 Separating Circle of  753  points:
```




```
Percentage of points inside the circle =  30.0531914893617
---------------
 Separating Circle of  1003  points:
```

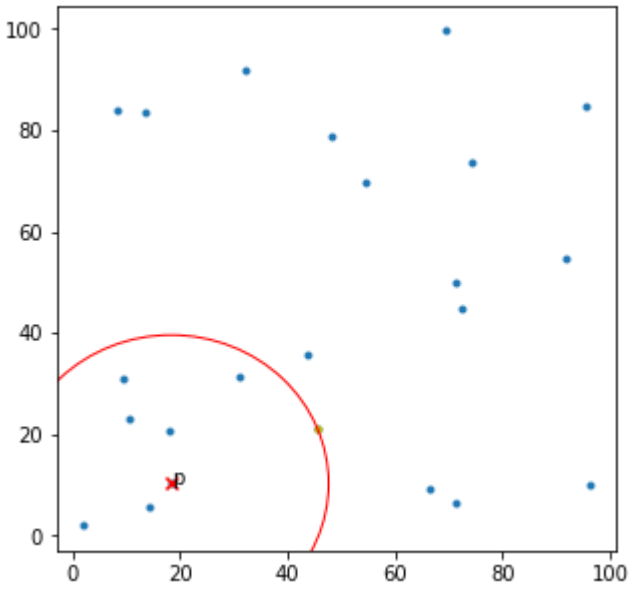



```
Percentage of points inside the circle =  29.599999999999998
```

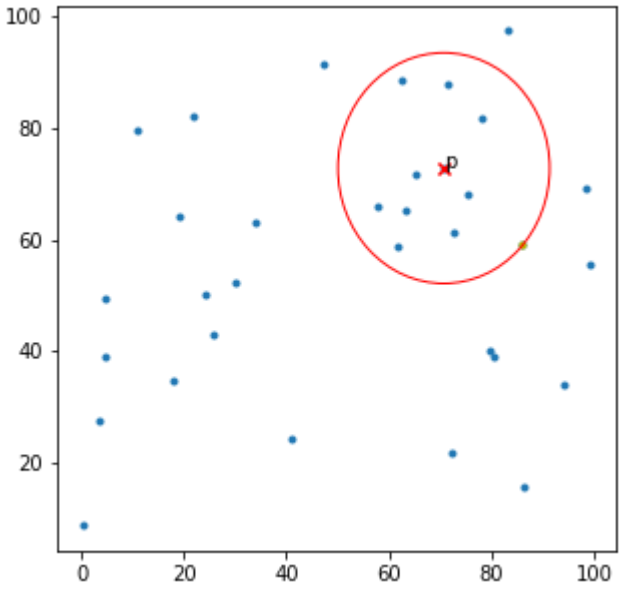
Outputs ::: Approach 2 ($\frac{3n}{10}^{th}$ smallest distance)

In [14]:
```
main_dist(2); main_dist(3); main_dist(4); main_dist(5); main_dist(10); main_di
st(50); main_dist(75); main_dist(100); # main(-1)
```

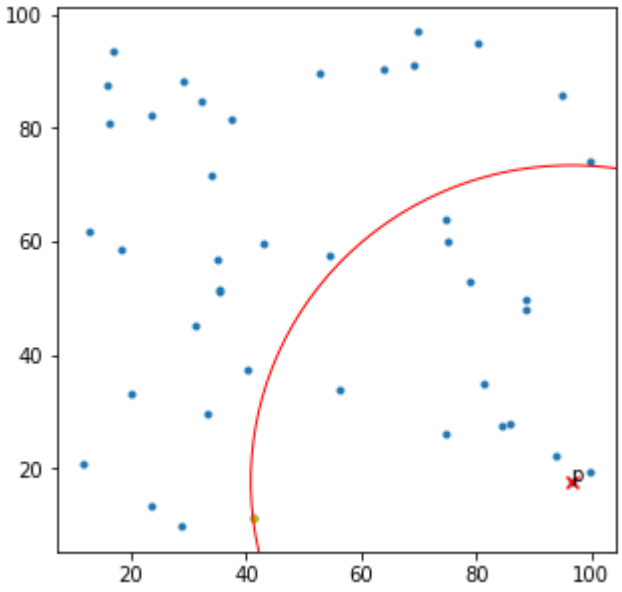
```
---------------
 Separating Circle of  23  points:
```



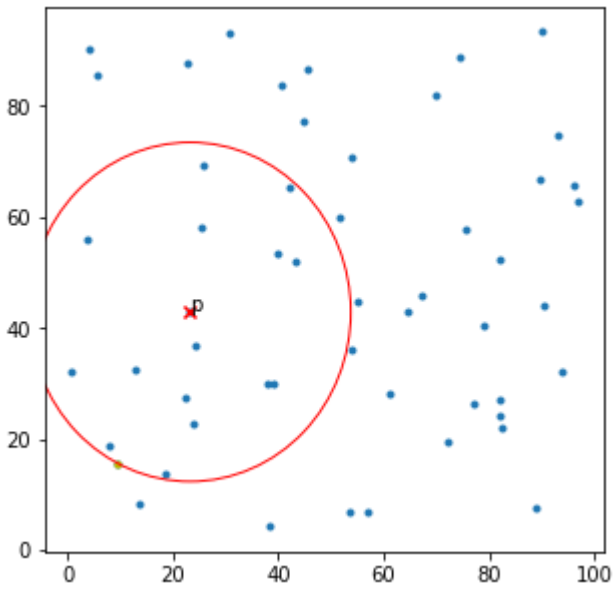

```
Percentage of points inside the circle =  31.818181818181817
---------------
 Separating Circle of  33  points:
```



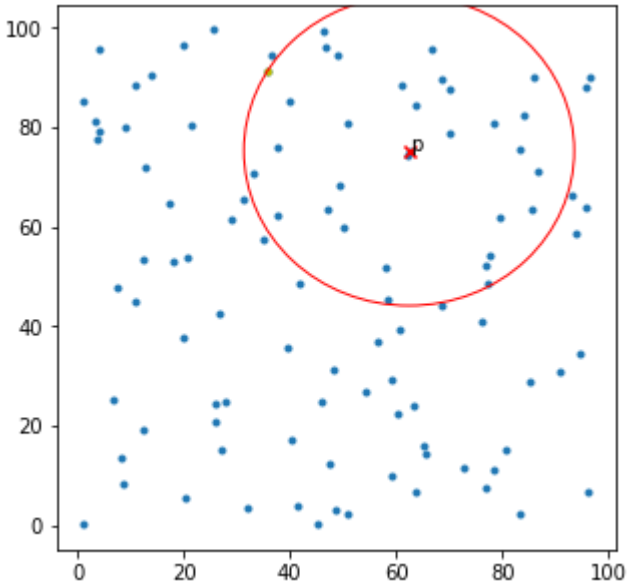

```
Percentage of points inside the circle =  31.25
---------------
 Separating Circle of  43  points:
```



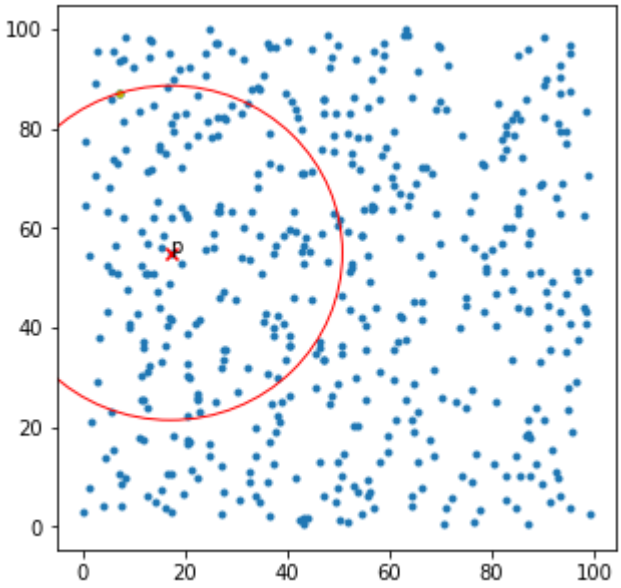

```
Percentage of points inside the circle =  30.952380952380953
---------------
 Separating Circle of  53  points:
```



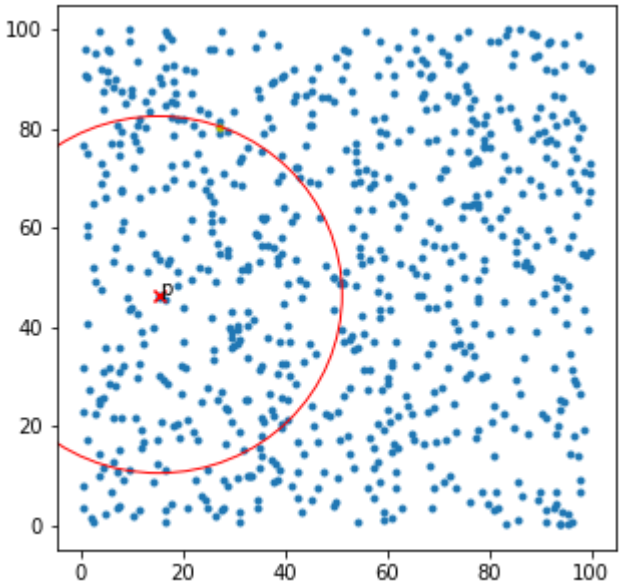

```
Percentage of points inside the circle =  30.76923076923077
---------------
 Separating Circle of  103  points:
```



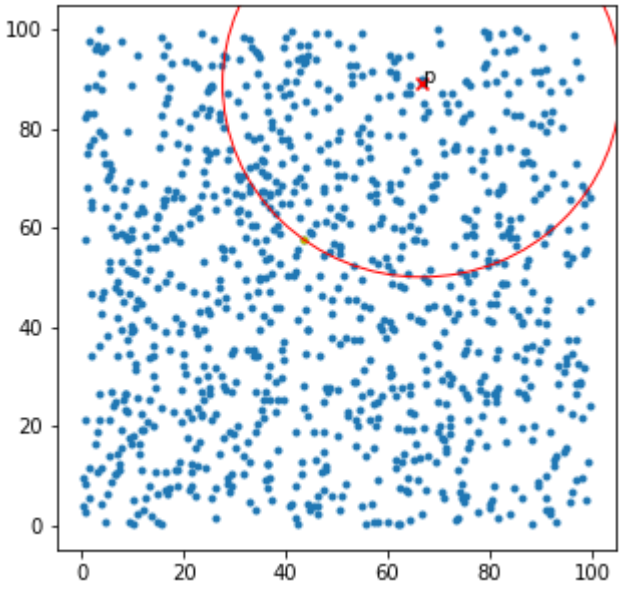

```
Percentage of points inside the circle =  30.392156862745097
---------------
 Separating Circle of  503  points:
```




```
Percentage of points inside the circle =  30.0796812749004
---------------
 Separating Circle of  753  points:
```




```
Percentage of points inside the circle =  30.0531914893617
---------------
 Separating Circle of  1003  points:
```




```
Percentage of points inside the circle =  30.039920159680637
```

Appendix : Actual Codes

Approach 1 ($\frac{3n}{10}^{th}$ order statistic angle)

- Sample n points in range $[0, 100] \times [0, 100]$
- Obtain points A, B and construct the line segment AB (`def getLineAB(pts, ax)`)
- Obtain all angles; $\angle AIB \; \forall I \in [n]$
- Sort the angles in decreasing order and obtain the point C with $\frac{3n}{10}^{th}$ largest angle.
- Obtain the circle with points A, B, C (`def get_Circle1(pts, pA, pB, ax)`)
- Plot the data and compute accuracy

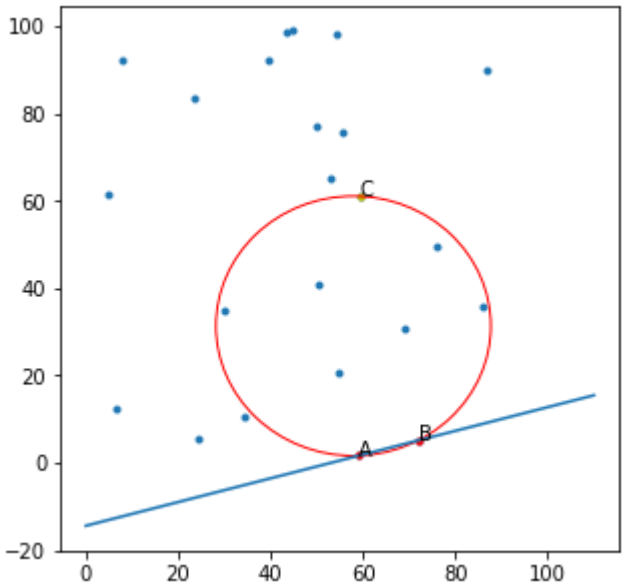
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import math
## -- helper functions
# compute the circle passing the given 3 points.
# if 3 pts are colinear, return radius = infinite (a degenerate circle)
def get_concircle(p1, p2, p3):
    temp = p2[0] * p2[0] + p2[1] * p2[1]
    bc = (p1[0] * p1[0] + p1[1] * p1[1] - temp) / 2
    cd = (temp - p3[0] * p3[0] - p3[1] * p3[1]) / 2
    det = (p1[0] - p2[0]) * (p2[1] - p3[1]) - (p2[0] - p3[0]) * (p1[1] - p2[1
])

    if abs(det) < 1.0e-6:
        return (None, inf)

    # Center of circle
    cx = (bc*(p2[1] - p3[1]) - cd*(p1[1] - p2[1])) / det
    cy = ((p1[0] - p2[0]) * cd - (p2[0] - p3[0]) * bc) / det

    radius = np.sqrt((cx - p1[0])**2 + (cy - p1[1])**2)

    return (cx, cy), radius

# returns randomly sampled int from [0,n-1] excluding pts in exclude[]
def distinct_sample(exclude, n):
    randInt = np.random.randint(0,n-1)
    return distinct_sample(exclude) if randInt in exclude else randInt

def getAngle(pA, pB, pC):
    CA = pC - pA
    CB = pC - pB
    dot = np.dot(CA,CB)
    det = CA[0]*CB[1]-CB[0]*CA[1]
    ACB = math.atan2(det, dot)

    # Converting to degree
    ACB = ACB * 180 / math.pi;
    if(ACB<0):
        ACB = 360+ACB
    return ACB

# gets a random line from set of points pts[]
def get_LineAB(pts, ax):
    n = pts.shape[0]
    # find 2 random pts to draw a line
    # for simplicity we consider two pts with lowest y coordinates
    yMin_indx = np.argpartition(pts[:,1], 2)
    pA = pts[yMin_indx[0]]
    pB = pts[yMin_indx[1]]
```

```
    # plotting the line b/w A and B
    m = (pA[1]-pB[1])/(pA[0]-pB[0])
    if(math.atan2(m, 1)<0):
        temp = pA
        pA = pB
        pB = temp
    c = -m*(pA[0])+pA[1]
    x = np.linspace(0, 110)
    ax.plot(x, m*x+c)

    # show pts A and B with labels
    ax.scatter([pA[0],pB[0]], [pA[1], pB[1]], c = 'r', marker = '.')
    ax.annotate("A", (pA[0], pA[1]))
    ax.annotate("B", (pB[0], pB[1]))

    return pA, pB, ax

# compute the required circle
def get_Circle1(pts, pA, pB, ax):
    n = pts.shape[0]
    angles = [] # angles subtended by all pts on line segment AB
    for i in pts: # compute angles
        if (i==pA).all() or (i==pB).all(): # ignore A and B
            continue
        else :
            gm = getAngle(pA, pB, i)

            angles.append((gm, i))

    angles.sort(reverse=True)

    pC = angles[int(3*n/10)][1] # obtain the separating point

    ax.scatter(pC[0], pC[1], c = 'y', marker = '.')
    ax.annotate("C", (pC[0], pC[1]))
    cntr, radius =get_concircle(pA, pB, pC) # compute a circle through A, B, C
    crcl = plt.Circle(cntr, radius, color='r', fill = False) # plot the circle
    ax.add_patch(crcl)

    return cntr, radius

# calculate the fraction of points inside the circle
def accuracy(cntr, radius, pts):
    n = pts.shape[0]
    inside_Circle = 0 # number of points inside the circle
    on_circle = 0 # number of points inside the circle, we'll ignore these poi
nts

    for i in pts:
        if np.linalg.norm(i-cntr) < radius:
            inside_Circle = inside_Circle+1
```

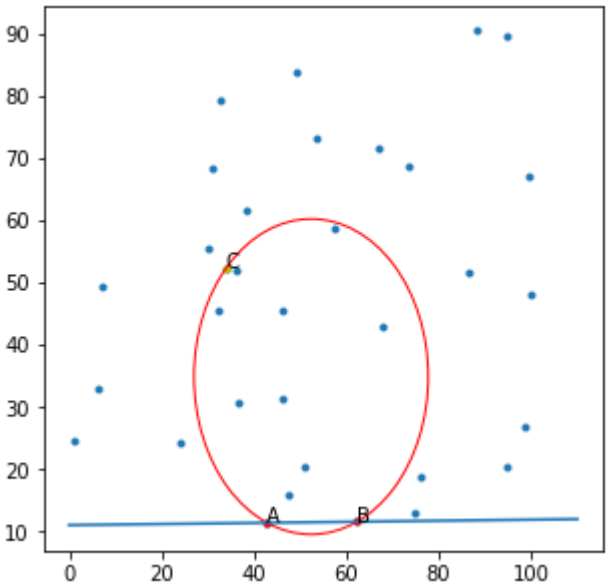
```
            if np.linalg.norm(i-cntr) == radius:
                on_circle = on_circle+1

    fraction_inside = inside_Circle/(n-on_circle)
    print("Percentage of points inside the circle = ", fraction_inside*100.0)
    pass

# computer the required circle,returns
def angle_Algorithm(pts, ax):
    pA, pB, ax = get_LineAB(pts, ax) # get the line b/w 2 random pts
    cntr, radius = get_Circle1(pts, pA, pB, ax)
    plt.show()

    accuracy(cntr, radius, pts) # lets verify
    return

## Build Dataset
def generate_data(r):
    if(r==-1):
        r = int(input("r = "))
    n = 10*r+3
    pts = np.random.sample((n,2))*100 # sample distinct pts

    # plot the points
    fig, ax = plt.subplots()
    fig.set_size_inches(5, 5)
    ax.scatter(pts[:,0],pts[:,1], marker = '.')
    return ax, pts
```

In [7]:
```
def main_angle(r):
    ax, pts = generate_data(r)
    print("---------------\n Separating Circle of ", pts.shape[0], " points:")
    angle_Algorithm(pts,ax)
    pass

main_angle(2); main_angle(3); main_angle(4); main_angle(5); main_angle(10); ma
in_angle(50); main_angle(75); main_angle(100); # main(-1)
```

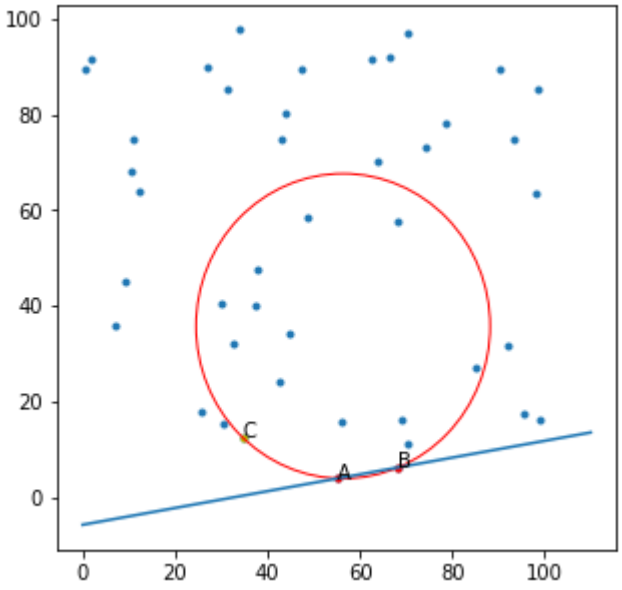
```
---------------
 Separating Circle of  23  points:
```



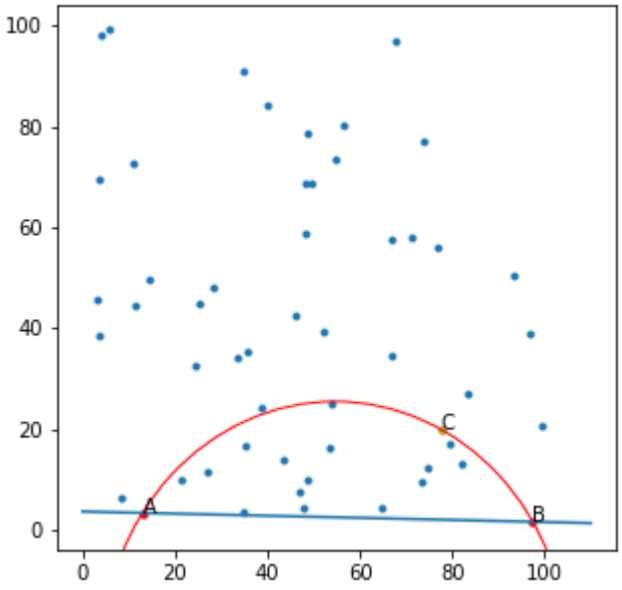

```
Percentage of points inside the circle =  27.27272727272727
---------------
 Separating Circle of  33  points:
```



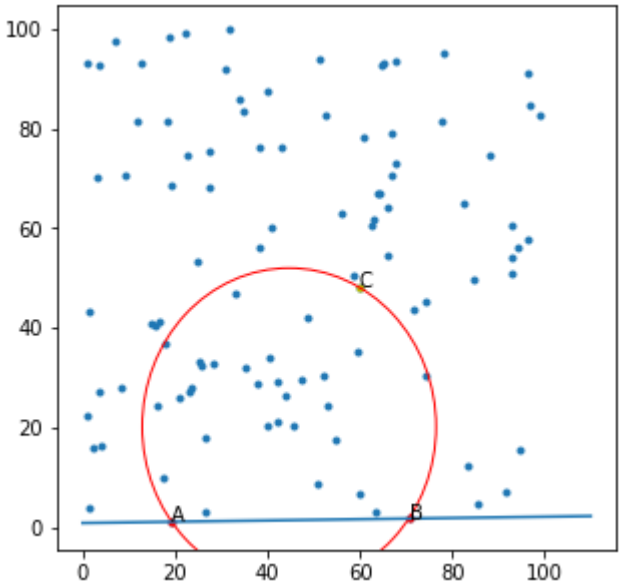

```
Percentage of points inside the circle =  29.03225806451613
---------------
 Separating Circle of  43  points:
```



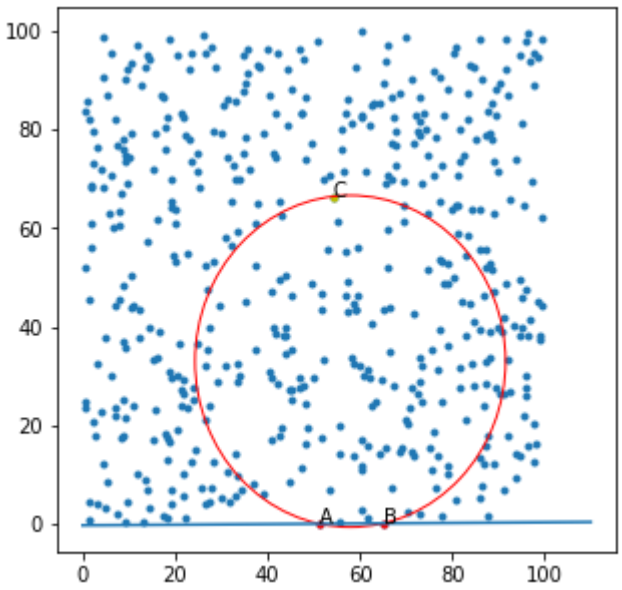

```
Percentage of points inside the circle =  30.952380952380953
---------------
 Separating Circle of  53  points:
```



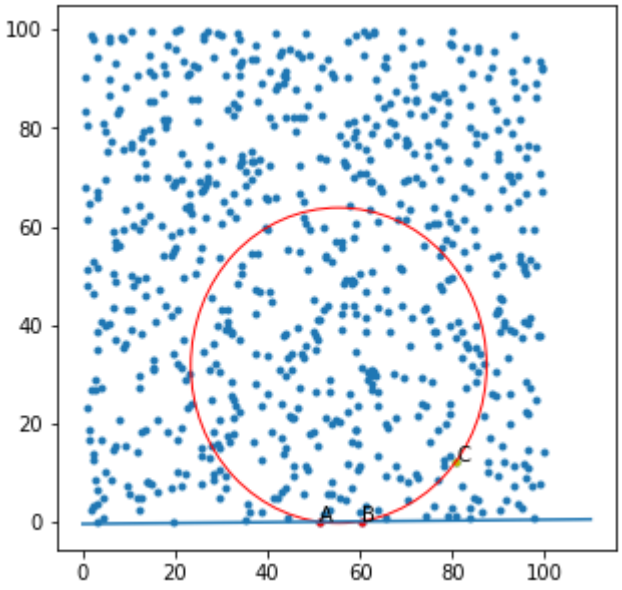

```
Percentage of points inside the circle =  30.0
---------------
 Separating Circle of  103  points:
```



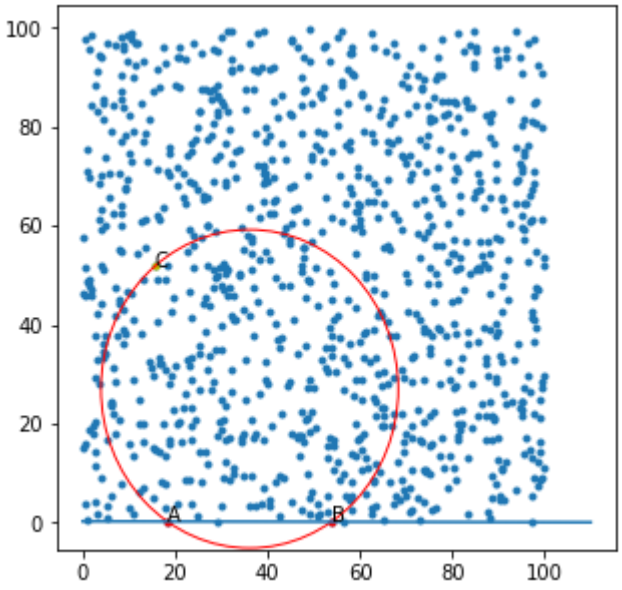

```
Percentage of points inside the circle =  31.372549019607842
---------------
 Separating Circle of  503  points:
```




```
Percentage of points inside the circle =  29.880478087649404
---------------
 Separating Circle of  753  points:
```




```
Percentage of points inside the circle =  30.186170212765955
---------------
 Separating Circle of  1003  points:
```




```
Percentage of points inside the circle =  29.97002997002997
```

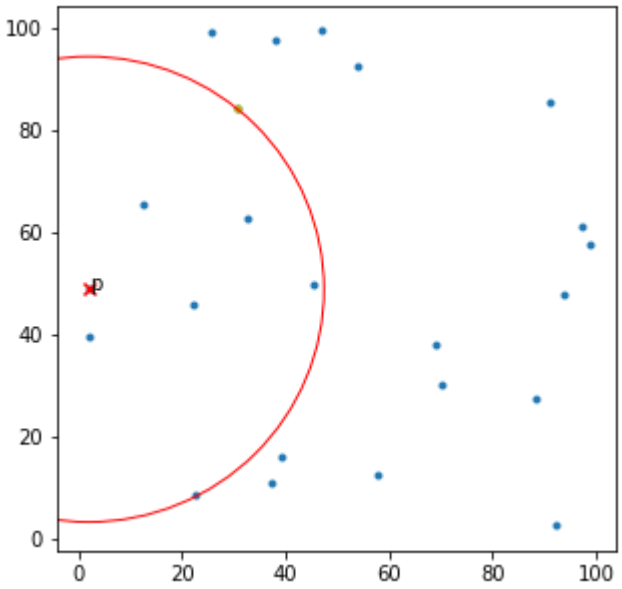
Approach 2 ($\frac{3n}{10}^{th}$ order statistic distance)

- Sample n points in range $[0, 100] \times [0, 100]$
- Sample the point $pCenter$ randomly
- Obtain all distances from $pCenter$; $dist(I, pCenter)\ \forall I \in [n]$
- Sort the distances in decreasing order and obtain the $\frac{3n}{10}^{th}$ smallest distance, say r.
- Obtain the circle with center $pCenter$, radius r (`def get_Circle(pts, ax)`)
- Plot the data and compute accuracy

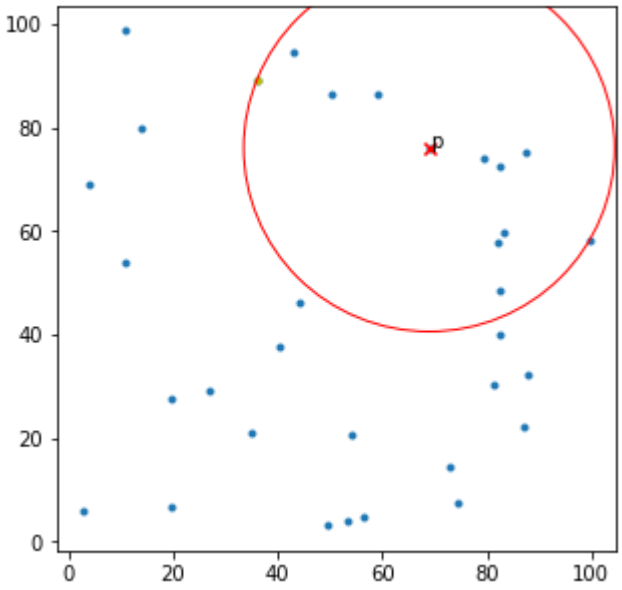
In [3]:
```
# compute the required circle
def get_Circle(pts, ax):
    n = pts.shape[0]
    pCenter =  pts[np.random.randint(0,n-1)]
    ax.scatter(pCenter[0], pCenter[1], c = 'r', marker = 'x')

    dists = [] # dist of  all pts from Center
    for i in pts: # compute distances
        if (i==pCenter).all(): # ignore the Center
            continue
        else :
            d = np.linalg.norm(pCenter-i)
            dists.append((d, i))
    dists.sort()

    pCircum = dists[int(3*n/10)][1] # obtain the separating point
    radius = dists[int(3*n/10)][0]

    ax.scatter(pCircum[0], pCircum[1], c = 'y', marker = '.')
    ax.annotate("p", (pCenter[0], pCenter[1]))
    crcl = plt.Circle(pCenter, radius, color='r', fill = False) # plot the cir
cle
    ax.add_patch(crcl)

    return pCenter, radius

# computer the required circle,returns
def dist_Algorithm(pts, ax):
    cntr, radius = get_Circle(pts, ax)
    plt.show()

    accuracy(cntr, radius, pts) # lets verify
    return
```

In [8]:
```
def main_dist(r):
    ax, pts = generate_data(r)
    print("---------------\n Separating Circle of ", pts.shape[0], " points:")
    dist_Algorithm(pts,ax)
    pass

main_dist(2); main_dist(3); main_dist(4); main_dist(5); main_dist(10); main_di
st(50); main_dist(75); main_dist(100); # main(-1)
```

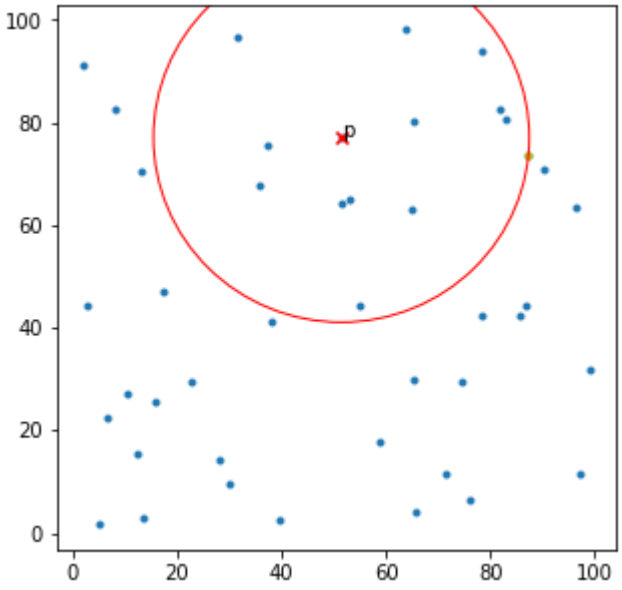
```
---------------
 Separating Circle of  23  points:
```



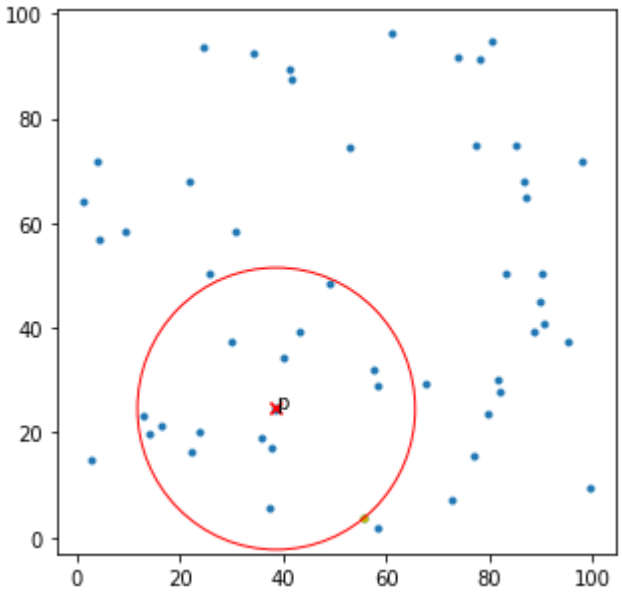

```
Percentage of points inside the circle =  31.818181818181817
---------------
 Separating Circle of  33  points:
```



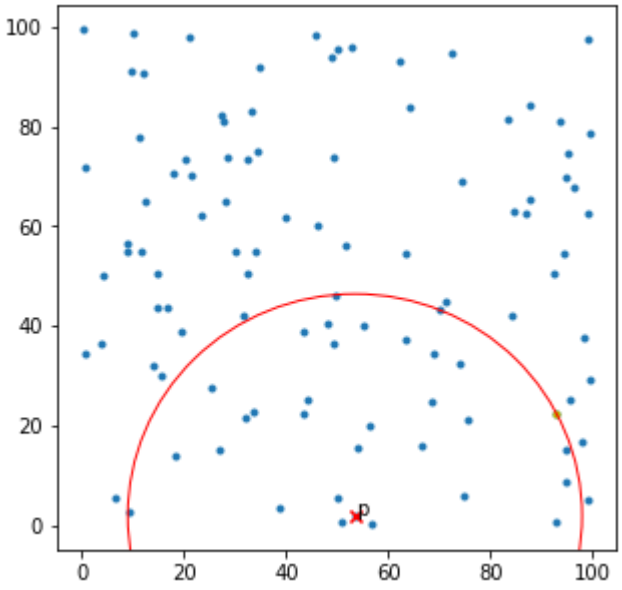

```
Percentage of points inside the circle =  31.25
---------------
 Separating Circle of  43  points:
```



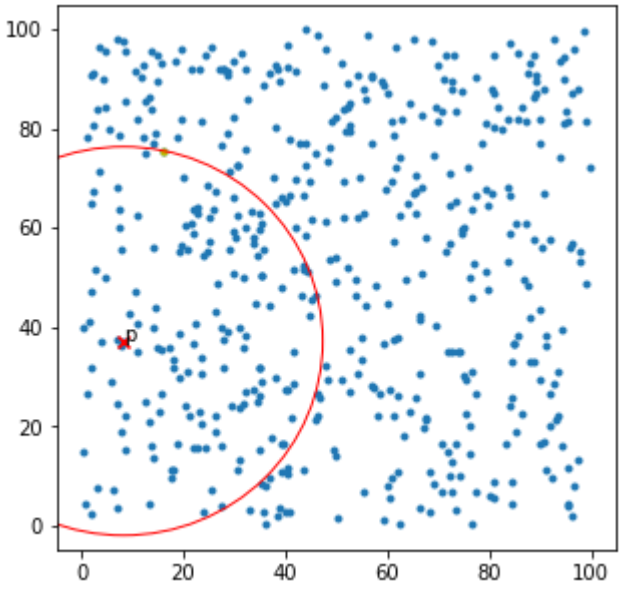

```
Percentage of points inside the circle =  30.952380952380953
---------------
 Separating Circle of  53  points:
```



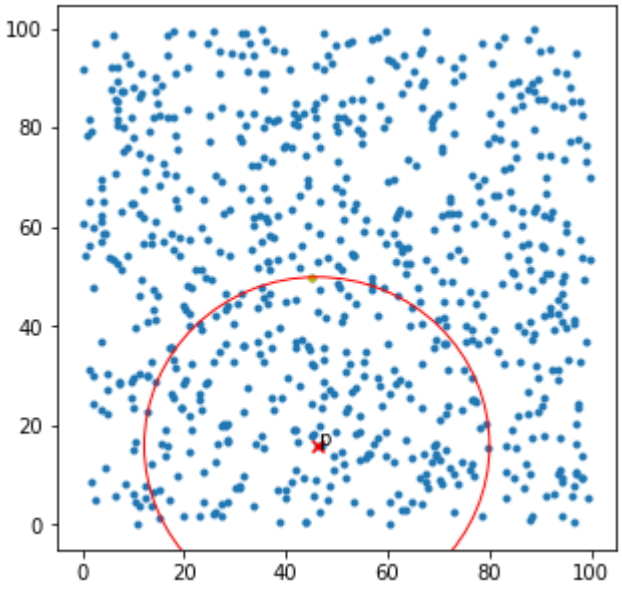

```
Percentage of points inside the circle =  30.76923076923077
---------------
 Separating Circle of  103  points:
```



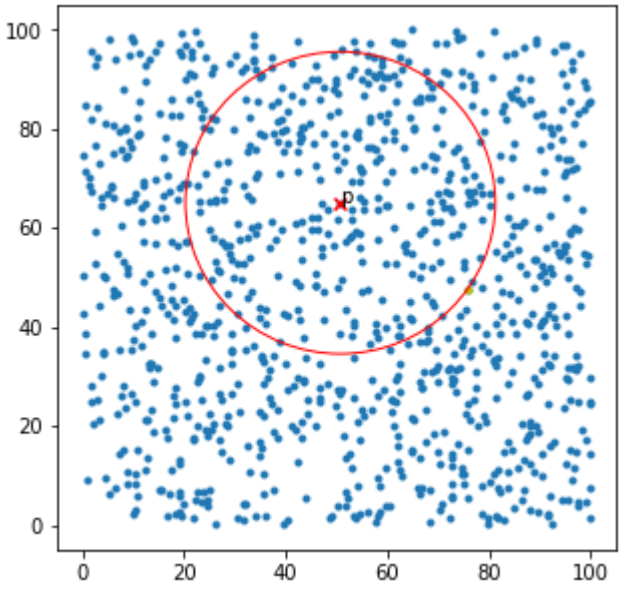

```
Percentage of points inside the circle =  30.392156862745097
---------------
 Separating Circle of  503  points:
```




```
Percentage of points inside the circle =  30.0796812749004
---------------
 Separating Circle of  753  points:
```




```
Percentage of points inside the circle =  30.0531914893617
---------------
 Separating Circle of  1003  points:
```




```
Percentage of points inside the circle =  30.039920159680637
```