Name-Balbhadra Tubid. Roll no.- CH22B040

1.) Determine the value of π using Monte-Carlo technique as described here. Choose a box of size 100x100 and draw a circle of known diameter inside it, perhaps using the centre of the square as the origin of the circle. Make sure that the circle is completely inside the square. Generate a pair of random numbers in the range 0 to 100 and use them as the coordinates of a point. If the point is located within the circle, increment the counter "inside". If the point is located outside the circle, increment the counter "outside". Repeat this a large number of times, say, about 1000 or so. Evaluate the ratio of the counters "inside" to "outside". If the random number generation was uniform within the range provided, then the ratio of the counters should be related to the ratio of the areas of circle and the square. Based on this numerical experiment, comment on the value of π you have obtained. Suggest what can be done to improve the accuracy of the value of π

- First I take initialize the diameter and diameter of circle.
- Then I run a loop a thousand times and initialize random values to x and y.
- Checking if the point is located within or outside the circle I increment the inside and outside values.

In [12]:
```
import random
import numpy as np

import matplotlib.pyplot as plt
```

In [13]:
```
number = 1000
x=np.arange(number)
y=np.arange(number)
centre=(50,50)
diameter= 80
inside=0
total=0
outside=0
```

In [14]:
```
for i in range(1000):
    x[i]= random.uniform(0,100)
    y[i]=random.uniform(0,100)
    distance=((x[i]-centre[0])**2 + (y[i]-centre[1])**2)**0.5
    if distance <= diameter/2 :
        inside=inside+1
    else:
        outside=outside+1
    total=total+1
```

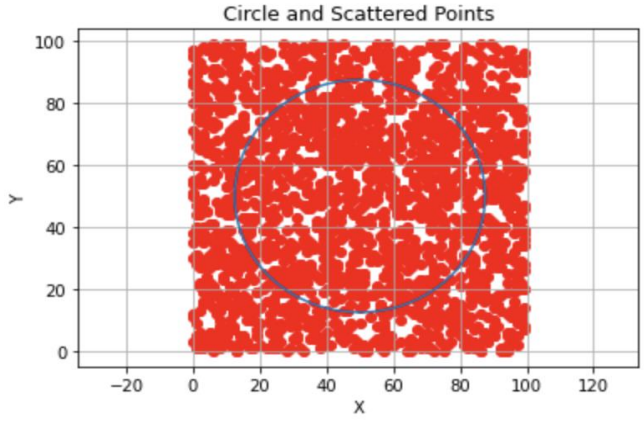
- I took out the ratio and multiplied it with 4 to get the value of pi,

- To get more accurate value of pi we should increase the number of times we are running the loop and taking circle with large diameter.

```
In [84]: theta = np.linspace(0, 2 * np.pi, 100)
         radius=diameter/2

         x_circle = centre[0] + radius * np.cos(theta)
         y_circle = centre[1] + radius * np.sin(theta)
         plt.plot(x_circle, y_circle,  label='Circle')

         plt.scatter(x, y, color='red', label=' Points')
         plt.axis('equal')
         plt.xlabel('X')
         plt.ylabel('Y')
         plt.title('Circle and Scattered Points')
         plt.grid()
         plt.show()
```




```
In [85]: ratio=inside/outside
         pie=ratio*4
         pie

Out[85]: 1772/557
```

2.) Create a 2 dimensional array T that holds floating numbers. Let the size of the array be, say, 100 along each dimension. Assume that the two indices of the array also represent the locations along two dimensions (x- and y- axes) where the value of the variable T is specified. Initialize the values of T such that it is 100 inside a circle of radius 25 and with an origin same as the centre of the square defined by the four corners (i,j) = { (1,1), (100,1), (1,100), (100,100)} assuming that the array index starts from 1. The values of T in the remaining region is 0. Use the following pseudo code to update the values of T for n steps (n can be say, 10, 50 and 100). That is, iterate the same instructions n times. Visualize the values of array T using a 2D plot such as pcolor or filled contours or surface plot. Comment on your observations. Pseudo code for each step: For i from 2 to 99 and j from 2 to 99: For j from 1 to 100: For i from 1 to 100: For your initial attemp, choose the value of λ to be 0.2 or lower. What happens if you choose a larger number? Make sure your report includes plots of T for initial condition and at least two different steps

- First I created 2-D array T of size 100x100 and updated its value to 100 wherever the the corresponding point's location of the array element is inside the circle.
- I drew then the contour plot of the T along with its corresponding (x,y) location.

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
```

```
In [2]: size=100
        size1=100-1
        T=np.zeros((size+1, size+1), dtype=float)
```

```
In [3]: centre = (size1/2, size1/2)
        radius = 25.0
        distance = 0.0
        centre
        print(T)
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

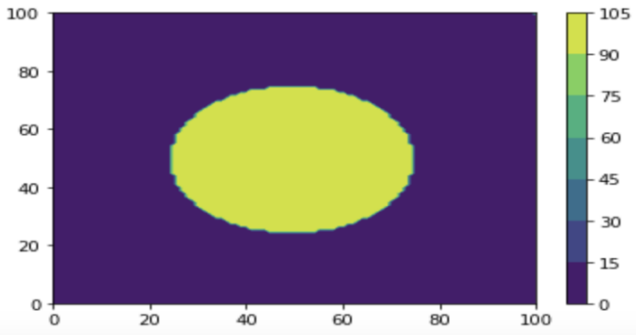
```
In [4]: for i in range(1,size+1):
            for j in range(1, size+1):
                distance=np.sqrt((i-centre[0])**2 +(j-centre[1])**2)
                #print(distance)
                if distance <= radius :
                    T[i,j] = 100.0
                    #print(distance)
                    #print(T[i,j])
                else :
                    T[i,j] = 0.0
```

- After this initialized a new array T new with its initial values same as T, I drew a contour lot for T.

```
In [5]: x=list(range(0,size+1))
        y=list(range(0,size+1))
        T[1,1]=5
        T[100,100]=78
        print(T)
```

```
[[ 0.  0.  0. ...  0.  0.  0.]
 [ 0.  5.  0. ...  0.  0.  0.]
 [ 0.  0.  0. ...  0.  0.  0.]
 ...
 [ 0.  0.  0. ...  0.  0.  0.]
 [ 0.  0.  0. ...  0.  0.  0.]
 [ 0.  0.  0. ...  0.  0. 78.]]
```

```
In [6]: plt.contourf(x,y,T)
        plt.colorbar()
        plt.show()
```

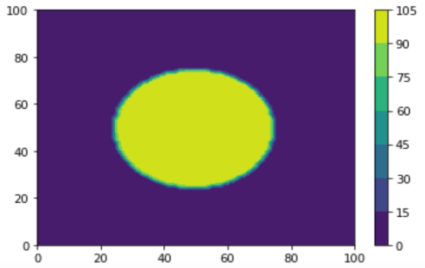


- Initializing a and n values I ran the given pseudo command to update the values of T_new.
- With the updated values of T_new I drew another contour plot, while corresponding with its (x,y) location.

In [19]:
```
n=100
a=0.2
```

In [20]:
```
for number in range(n):
    T_new = np.copy(T)
    for i in range(2,size):
        for j in range(2,size):
            T_new[i,j]= T[i,j] +a*(T_new[i+1,j] + T_new[i-1,j] + T_new[i,j+1] + T_new[i,j-1] -4*T_new[i,j])
    for j in range(1, size+1):
        T_new[1,j] = T_new[2,j]
        T_new[100,j] = T_new[99,j]
    for i in range(1,100):
        T_new[i,1] = T_new[i,2]
        T_new[1,100] = T_new[i,99]
```

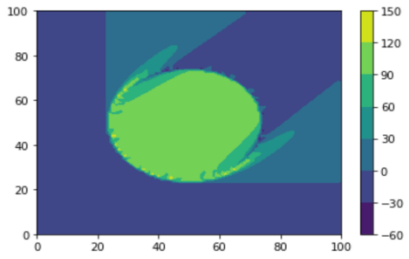
In [21]:
```
plt.contourf(x,y,T_new)
plt.colorbar()
plt.show()
```



In [7]:
```
n=100
a=0.5
```

In [8]:
```
for number in range(n):
    T_new = np.copy(T)
    for i in range(2,size):
        for j in range(2,size):
            T_new[i,j]= T[i,j] +a*(T_new[i+1,j] + T_new[i-1,j] + T_new[i,j+1] + T_new[i,j-1] -4*T_new[i,j])
    for j in range(1, size+1):
        T_new[1,j] = T_new[2,j]
        T_new[100,j] = T_new[99,j]
    for i in range(1,100):
        T_new[i,1] = T_new[i,2]
        T_new[1,100] = T_new[i,99]
```

In [9]:
```
plt.contourf(x,y,T_new)
plt.colorbar()
plt.show()
```

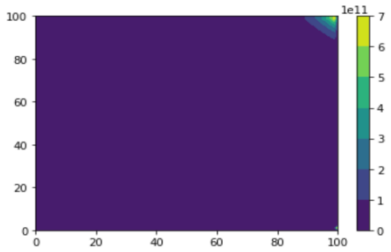


- The contour plot of T_new does not change much with the n values but changes drastically with the increment of values.

In [11]:
```
n=100
a=0.6
```

In [15]:
```
for number in range(n):
    T_new = np.copy(T)
    for i in range(2,size):
        for j in range(2,size):
            T_new[i,j]= T[i,j] +a*(T_new[i+1,j] + T_new[i-1,j] + T_new[i,j+1] + T_new[i,j-1] -4*T_new[i,j])
    for j in range(1, size+1):
        T_new[1,j] = T_new[2,j]
        T_new[100,j] = T_new[99,j]
    for i in range(1,100):
        T_new[i,1] = T_new[i,2]
        T_new[1,100] = T_new[i,99]
```

In [16]:
```
plt.contourf(x,y,T_new)
plt.colorbar()
plt.show()
```



- By choosing a larger a value, the rate at which the values in T change will be faster. This means that the heat or diffusion process will occur more rapidly in each step. Consequently, the propagation of values from the high-concentration region to the surrounding area will be more pronounced.