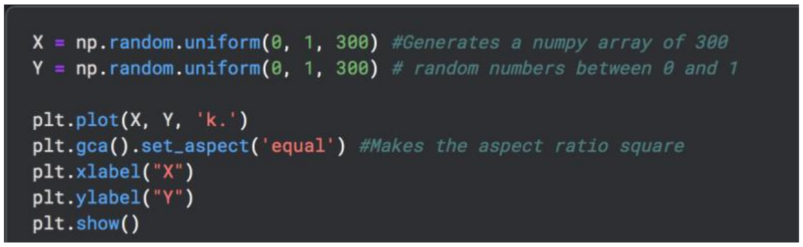
Calculating pi with Random Numbers:

Background:

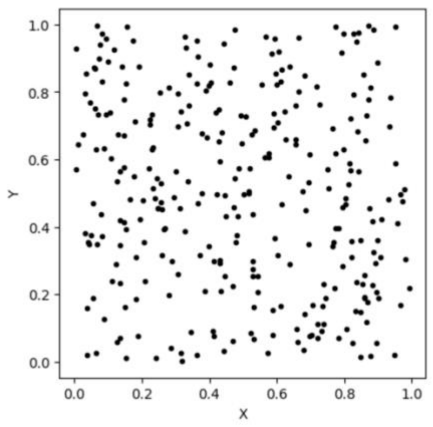
Random number generators generate a random number from a chosen probability distribution function. However, one can only choose from a limited number of distributions. This requires some cleverness when designing an algorithm. For example, if we want to generate a random number between 0 and 1, we can just use an existing function from the numpy library: np.random.uniform(0, 1, 300), which would generate an array of 300 random numbers between 0 and 1. However, if we want to generate a uniform set of points in a unit circle, things are not as straight forward. To see why, consider the following:

If we want to generate a uniform distribution of points inside the unit square (defined as $X \in [0,1]$ & $Y \in [0,1]$), we can simply generate an X and Y coordinate independently.

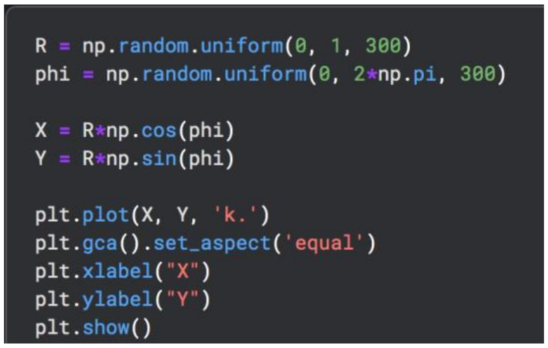


```
X = np.random.uniform(0, 1, 300) #Generates a numpy array of 300
Y = np.random.uniform(0, 1, 300) # random numbers between 0 and 1

plt.plot(X, Y, 'k.')
plt.gca().set_aspect('equal') #Makes the aspect ratio square
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```



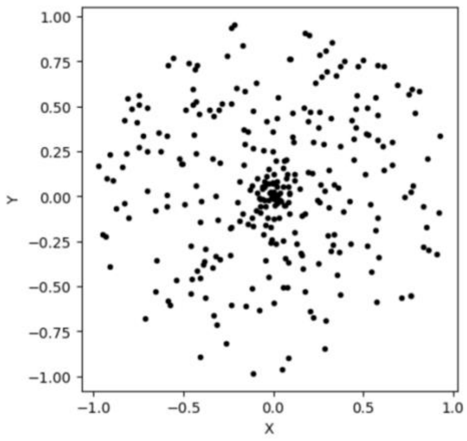
But what if we repeat this for points inside the unit circle (R < 1)? We might expect the same process to work in polar coordinates…



```
R = np.random.uniform(0, 1, 300)
phi = np.random.uniform(0, 2*np.pi, 300)

X = R*np.cos(phi)
Y = R*np.sin(phi)

plt.plot(X, Y, 'k.')
plt.gca().set_aspect('equal')
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```



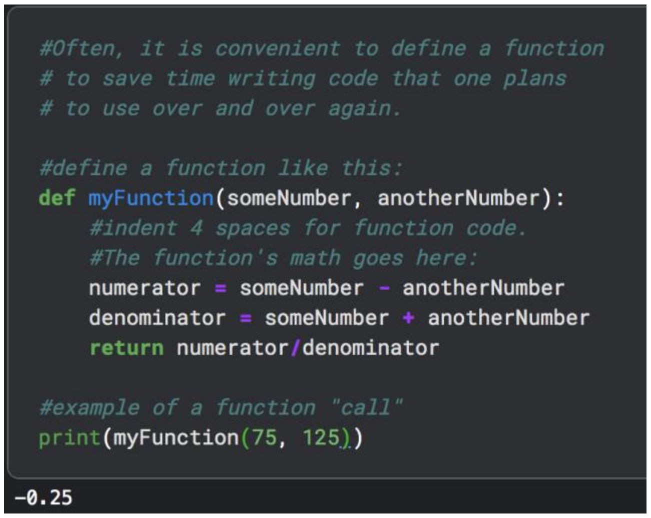
…but the resulting distribution is definitely not uniform! The points are too concentrated in the center. To see why, consider the code we used to generate this set of points. We randomly selected a radial coordinate between 0 and 1 and randomly chose an angle between 0 and 2π.

But there is more area in the outer reaches of the circle, so if we want a uniform distribution of points, we need to distribute more points at large radii. There is often no built-in number generator for purposes like this one, so we can instead be clever about how we design our algorithm. This is your goal for today's lab.

Skills:

First, we'll need to learn a few new things about python syntax:

- Random Numbers (included above!)
- Functions – a block of reusable code:



```
#Often, it is convenient to define a function
# to save time writing code that one plans
# to use over and over again.

#define a function like this:
def myFunction(someNumber, anotherNumber):
    #indent 4 spaces for function code.
    #The function's math goes here:
    numerator = someNumber - anotherNumber
    denominator = someNumber + anotherNumber
    return numerator/denominator

#example of a function "call"
print(myFunction(75, 125))
```

```
-0.25
```

Problems:

1. A better way to generate a uniformly distributed set of points in the unit circle is to build the following algorithm:
    a. Randomly generate an X and Y coordinate from a uniform probability distribution on the interval -1 to 1.
    b. If the point is inside the unit circle, keep it. If the point is outside the unit circle, reject it.
    c. Repeat until a sufficient number of points are acquired.

    Write an algorithm that generates a uniform distribution of N = 500 points inside the unit circle. Plot the results.

2. Now let's write another version of this code in a new cell. This time, let's just focus on the first quadrant. Generate an X and Y coordinate from a uniform distribution – each on [0, 1] – and keep track of the total number of points generated and the total number accepted. Start out by generating N_{tot} = 1,000 points. Use this to estimate pi.

3. How does your estimate of pi change when N_{tot} = 10,000 points? What about 100,000 points? What percent error do you find compared to numpy's built in value of pi: `np.pi`?

4. Now let's switch gears a bit to functions.
   a. Define a function that takes two arguments: an array of X coordinates and an array of Y coordinates. Have this function return the average (2D) distance amongst all of the points passed to the function as an argument.

   b. Then, create 1,000 random points distributed uniformly over a unit square.

   c. Use the function on these points to explore the average separation distance of uniformly distributed points in a unit square. (We can actually do this integral analytically and find that the answer is $\frac{1}{15}\left(2+\sqrt{2}+5\ln\left(1+\sqrt{2}\right)\right) \approx 0.5214\ldots$ Do your numerical results agree?)