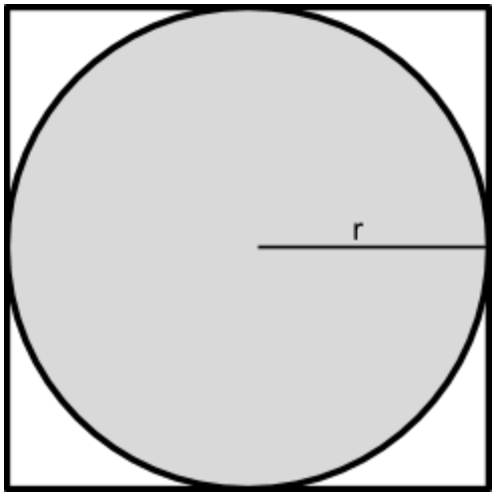
Monte Carlo Simulation: Calculating π

Background

In section today we will try to use random numbers to calculate π. To understand how a precise mathematical constant can be estimated using randomness we first need to refresh some basic geometry.

In the figure on the right the circle and the square have the following areas:



Area Circle: $A_C = \pi r^2$
Area Square: $A_S = (2r)^2 = 4r^2$

The ratio of the two areas is: $\frac{A_C}{A_S} = \frac{\pi r^2}{4r^2}$

And solving for π we get: $\pi = \frac{4A_C}{A_S}$

If we have an estimate for the ratio of the area of the circle to the to the square we can solve for pi. The challenge becomes estimating this ratio.

Monte Carlo Simulations

This is where we can take advantage how quickly a computer can generate pseudorandom numbers. There is a whole class of algorithms called Monte Carlo simulations that exploit randomness to estimate real world scenarios that would otherwise be difficult to explicitly calculate. We can use a Monte Carlo simulation to estimate the area ratio of the circle to the square.

Imagine you randomly drop grains of sand into the area of the square. By counting the total number of sand grains in the square (all of them since you're an accurate dropper) to the number of sand grains inside the circle we get this estimate. Multiple the estimated ratio by four and you get an estimate for π. The more sand grains you use the more accurate your estimate of π.

Today's Problem

Write a program that estimates π for $[10^0,10^1,10^2,10^3,10^4,10^5,10^6,10^7,10^8]$ random points. Your output should look something like this:

```
1x10^0: 4.0
1x10^1: 2.4
1x10^2: 2.68
1x10^3: 3.004
1x10^4: 3.1296
1x10^5: 3.14212
1x10^6: 3.14442
1x10^7: 3.1415972
1x10^8: 3.14182688
```

Step 1: Plan It Out

What are some major design considerations you need to think about? How the problem be broken into a series of smaller problems?

What are some useful functions to create? For each function write a doc string.

Hint: From the random modual: random.random() returns a float between 0 and 1.

Step 2: Implement the smaller problems.

Step 3: Write the main function.

Bonus Question: How would you implement this as a monolithic function instead of multiple functions? What are the pros and cons of each approach?

Monte Carlo Simulation: Calculating π
(Solutions)

Step 1: Plan It Out

Design Considerations:

- How general to make each function? Can the radius of the circle be changed?
- Where to center the square in the coordinate plane?

Some useful functions might include:

- random_around_zero(r): return a random float between [-r,r]
- random_throw(r): return the tuple (x,y) of a random point inside a square [(-r,r),(r,r),(r,-r),(-r,-r)]
- in_circle(r,x,y): return True if the point (x,y) is in a circle of radius r centered at (0,0). Return False otherwise.
- number_in_circle(n_total,r=0.5): test n_total number points and return the number that are in a circle of radius r (default r=0.5)

Step 2: Implement the smaller problems.

```
def random_around_zero(r):
    # Return a random float in the range [-r,r]
    width = 2*r
    return random.random()*width - r

def random_throw(r):
    #Retrun a random point in the square [(-r,r),(r,r),(r,-r),(-r,-r)
    x = random_around_zero(r)
    y = random_around_zero(r)
    return x,y

def in_circle(r,x,y):
    #Return True if (x,y) is in a circle with radius r centered at 0. Otherwise False.
    d = (x*x + y*y)**0.5
    return d<=r

def number_in_circle(n_total,r=0.5):
    '''Test n_total random points (x,y) in a square with sides 2r and return the number
    of points that are inside a circle of radius r.'''
    n_circle = 0
    for i in xrange(n_total):
        x,y = random_throw(r)
        if in_circle(r,x,y):
            n_circle += 1
    return n_circle
```

Step 3: Write the main function.

```
def main():
    for i in range(9):
        n_total = 10**i
        n_circle += number_in_circle(n_total)
        pi = 4.0 * n_circle/n_total
        print '1x10^' + str(i )+ ':',pi

if __name__ == '__main__':
    main()
```

A monolithic approach:

Instead of breaking the problem down into smaller functions you could have written a single function that calculates the number of throws in the circle without any helper functions. What are some pros and cons of designing it this way vs the functional decomposition method?

```
def number_in_circle_monolithic(n_total,r=0.5):
    n_circle = 0
    for i in xrange(n_total):
        x = 2*r*random.random() - r
        y = 2*r*random.random() - r
        d =( x*x + y*y)**0.5
        if d<=r:
            n_circle += 1
    return n_circle
```