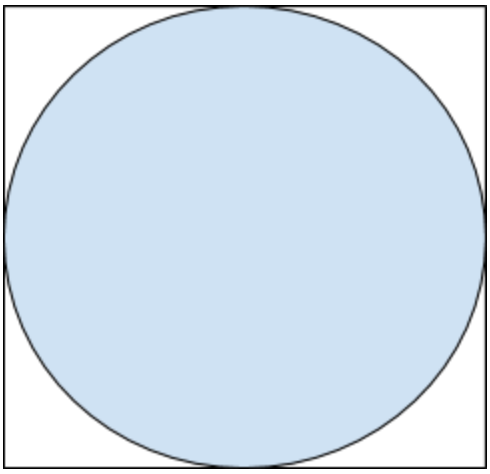
Introduction to Monte Carlo Simulations: Estimating the Value of Pi

We know that the value of π is approximately 3.141592.... There are many ways of getting this estimate. One way involves a combination of geometry, probability, and Monte Carlo simulations.

For this exercise you will need:

- Pencil
- Calculator
- Mathematica



The image above shows a circle drawn inside of a square. The radius of the circle is 1 unit. Imagine this is a dartboard, and you randomly throw a dart at the board.

1) **Estimate** the probability that your dart will hit the shaded section. Is the probability 0? ½? 1? Somewhere in between? Discuss your answer with your group and explain your reasoning.
   The probability is greater than ½ and less than 1. We can tell by dividing the square into 4 squares and seeing that the circle extends over a diagonal line drawn through the middle of each square.

2) Using what you know about area and probability, check your answer from 1) by calculating the **exact** probability that your dart will hit the shaded section. Do not round your answer.
   $\pi/4$

We can use Monte Carlo simulations to estimate the value of π by randomly picking points within the dartboard and finding the fraction of points that fall inside the shaded circle and comparing it to the actual probability found in 2).

3) We will write a Mathematica program to do this. Before writing our program, though, let's imagine what our dartboard would look like in the xy-coordinate plane. The board (and the circle) will be centered at the origin and will extend from -1 to 1 in both the x and y directions. What is the equation of this circle?
    a) Open Mathematica and create a new notebook.

    b) Save the notebook as "EstimatePi."

    c) First we will want to generate some random coordinate points that fall inside our square. First we want to tell Mathematica how many points we would like. For now, let's make it 10. Type: **pts = 10;**

    d) We want to generate two lists of numbers. One will be the x-coordinates for our points, and the other will be the y-coordinates. We want Mathematica to randomly choose any number from -1 to 1 for the two lists, so we will use the RandomReal function:

    X = RandomReal[{-1,1}, pts];
    Y = RandomReal[{-1,1}, pts];

    The **{-1,1}** tells Mathematica the range to pick numbers from, and **pts** tells Mathematica how many numbers to pick.

    Run these statements without the ending semicolons 2 or 3 times to see what you get. The numbers should change each time. It is possible to get Mathematica to choose the same random numbers each time by using something called a *seed*.

    To do this, delete your other lines and type instead:

    SeedRandom[1234]
    X = RandomReal[{-1,1}, pts];
    Y = RandomReal[{-1,1}, pts];

    Seeds are important for several reasons, one being that other people will be able to get the same results that you do in your numerical experiment. Just like scientists want to give people instructions for experiments so that other people will be able to perform them and get the same result, we also want other mathematicians or computer scientists to be able to replicate what we get by using the same seed even though the numbers are random.

e) In order to "access" one of the numbers in a list, you need to type the list's name and then use double brackets around the term number. For example, try typing **X[[2]]** and run the statement. You should get the second number in the **X** list. You can delete this statement now.

f) Now we want to count the number of points that fall within the circle and then compare that to the total number of points. To do this, we can test each point, and if it falls within the circle, assign it the number 1. If it falls outside the circle, we can assign it the number 0. Then we can add the 1s to see how many points are in the circle.

To do this, we'll create a table for the values and name it **HitMiss**. If we take the coordinates of the point, substitute them into $x^2 + y^2$, and the value is less than the radius of the circle squared, then we can assign that point a 1 otherwise we give it a 0.
If students have trouble understanding why this tells whether or not the point is inside the circle, you can explain it to them using the Pythagorean Theorem.

```
HitMiss = Table[
If[X[[i]]^2 + Y[[i]]^2 < 1, 1, 0],
{i, 1, pts}
];
```

This statement is creating a table (in this case it is a list) with 10 spaces (or whatever number we assign **pts** to be). The **If** statement inside the table is testing each value of **X** and **Y** to see if the point is in the circle or not. It starts from **i = 1**, testing the first values, then 2, testing the second values, etc., all the way to **i = pts** (which we know to be 10).

Try running the statement without the ending semicolon to see the result.

g) To find how many hits, we can sum up the numbers in the **HitMiss** table:
Hits = Total[HitMiss];

h) The exact probability of hitting the shaded section is $\pi/4$. (This would be if we threw an infinite number of darts!). So the number of hits divided by the total number of points should be *approximately* $\pi/4$. If we multiply this by 4, then it should be *approximately* π.
ApproxPi = 4*Hits/pts

Try running this statement and see what you get. You might notice that Mathematica gives you a fraction instead of a decimal. It would be nice to have a

decimal so we could see if it is close to π without having to do extra calculations.

We need to give Mathematica some hint that we would like to see a decimal instead of a fraction. To do this change the statement you just typed to

ApproxPi = 4.0*Hits/pts

Run the statement again. Your answer should be a decimal this time.

We will add a little more to our program in a bit. Before we do, let's discuss something very important in the field of computation: Error! What is error? Error is the difference between our calculated answer and the actual answer. There are different ways of measuring this, but we would like to see the error between our Monte Carlo approximation of π and the actual value of π.

Visit the following website, and read section 3 about both *absolute error* and *relative error*:
http://www.regentsprep.org/regents/math/algebra/am3/LError.htm
Discuss what you learned with your group members.

i) Let's find both the absolute and relative error for our approximation of π.
AbsErr = Abs[ApproxPi - Pi]

Discuss with your group members how to program relative error and come up with your own statement. Use the website and the statement above to help you.
Name your relative error: **RelErr**
This line should be **RelErr = Abs[ApproxPi - Pi]/Pi**

4) Let's take a look at what happens when we choose greater numbers of points. What would you expect to happen?

These may not be exactly the same values.

pts	ApproxPi	AbsErr	RelErr
10	3.2	0.0584073	0.0185916
100	3.08	0.0615927	0.0196056
1000	3.12	0.0215927	0.00687316
10000	3.1536	0.0120073	0.00382206
100000	3.15116	0.00956735	0.00304538

1000000	3.14537	0.00377935	0.00120
10000000	3.14165	0.0000573464	0.0000182539

5) What observations do you have about your results? Do they match your predictions? Why or why not?
It's surprising that the ten point prediction is slightly more accurate than the 100-point prediction, but this might be different for a different set of random points. In general, the accuracy of the estimation of pi increases as the number of points increases.

6) Take a look at the level of accuracy for each number of **pts**. How many decimal points of your approximation match their actual value in π. Discuss possible disadvantages of using this method to estimate π.
We really only get 2 decimal places of accuracy with 10,000,000 points, which means that to get a more accurate estimation, we would have to have an ENORMOUS number of points. This is a lot of work for very little yield. May be good to mention to the students that using greater and greater numbers of points will make the program increasingly slower to run.

7) If you would like to visualize this experiment, add the following statements to your program:

```
pairs = Table[{X[[i]], Y[[i]]}, {i, 1, pts}];
Show[Graphics[{Lighter[Red], Disk[{0, 0}, 1]}], ListPlot[pairs]]
```

Run these two statements for different values of **pts** to see the distribution of points. Discuss with your group members what you think the different parts of these statements mean.
This is open-ended. The pairs statements create coordinate pairs. The show statement creates a red circle centered at (0,0) with radius 1 and **ListPlot** actually plots the the coordinate points.