# Computing Pi

Sarah Mann

April 12, 2012

In this assignment, you will compute the value of $\pi$ using a random number generator and basic arithmetic. You will write one program to do this, which you should call `pi.py`. You will write this program in a few stages and will turn in one or more of these stages before your final program is due. You will also answer some questions related to this program and turn these in as well.

## 1 Circles in Squares

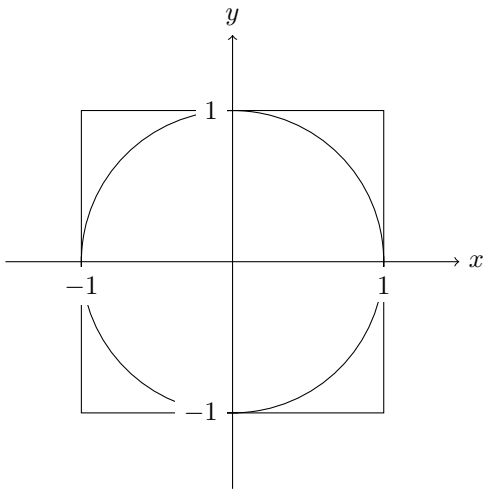


Figure 1: This figure shows a square centered on the origin of a coordinate axis. The square's corners are located at $(1, 1), (1, -1), (-1, -1)$, and $(-1, 1)$. Inside this square is a circle, also centered at the origin, with radius 1.

**Question 1**: What is the area of the square depicted in Figure **??**?

**Question 2**: What is the area of the circle depicted in Figure **??**?

**Question 3**: If you randomly chose a point inside the square in Figure **??**, what is the probability that point will also be inside the circle?

## 2 Sketch of Program

The analysis of Figure **??** suggests a method for computing $\pi$. If we randomly choose points inside the square, then $\frac{\pi}{4}$ of those points should be inside the circle as well. Thus we can compute $\pi$ in the following way:

1. Let `n` be the number of iterations to run, and `InCircle` be the number of points that were inside the circle
2. Do the following things `n` times:
   (a) Randomly select a point `(x,y)` inside the square
   (b) If `(x,y)` is also inside the circle then:
      i. Add one to `InCircle`
3. Compute `ApproxPi`, our approximation of $\pi$ from `InCircle` and $n$
4. Compare `ApproxPi` to $\pi$

Your goal is to write a program to do this. The rest of this document discusses how to do each of these steps.

## 3 Selecting a point in the square

We have previously used the `random` library and `random.randrange` to generate random integers. The `random` library has some other useful functions; today we will use `random.uniform(a,b)` to generate random floats. `random.uniform(a,b)` accepts two inputs `a` and `b` and returns a randomly selected float in the range `(a,b)`. Here is an example which you should run from your interactive window:

```
>>> import random
>>> random.uniform(0,1) # try this line a few times
```

What do you see?

### 3.1 Program Stage 1

Write a short program that randomly chooses a point `(x,y)` inside the square in Figure **??**, and prints that point.

## 4 Is the point inside the circle?

**Question 4**: Give the formula for the distance between a point $(x, y)$ and the origin.

**Question 5**: Given two values `x` and `y`, how can you test (in Python!) if the point `(x,y)` is inside the circle in Figure **??**. Your answer should include one line of Python code that completes this task, as well as an explanation of your code.

### 4.1 Program Stage 2

Building on your program for Stage 1, write a program that randomly chooses a point `(x,y)` inside the square in Figure **??**, prints that point, then determines if the point is inside the circle in Figure **??** and prints an appropriate message. Here are two examples of what happens when I run my program:

```
(0.4066222280242917, 0.1357168750269946)
The point is in the circle.
```

```
(0.8176168111658206, -0.9989834077422652)
The point is not in the circle.
```

## 5 Do it $n$ times

Now that we can randomly pick a point in the square and test if it is in the circle, we need to do this a lot of times and keep track of how often the point is in the circle. This will be stage 3 of your programming assignment.

### 5.1 Program Stage 3

Building on your program for Stage 2, write a program that:

1. Asks the user how many points she wants to choose (we'll call this number $n$)
2. Does the following $n$ `n` times:
   (a) Randomly selects a point `(x,y)` inside the square
   (b) Checks if `(x,y)` is also inside the circle and counts how often this happens
3. Computes an approximation of $\pi$:
   (approximation of $\pi$) = $4\times$ (number of times the point was inside the circle)/n
4. Prints $n$, the number of times the point was inside the circle, and the approximation of $\pi$

## 6 Comparing our approximation of $\pi$ and $\pi$

As the final piece to this project, we would like to know how good our approximation to $\pi$ is. Recall that the `math` library contains a numerical value for $\pi$, `math.pi`. We will compare our approximation to this value. Create a new variable called `Error` by adding this line of code to your program:

```
Error = ApproxPi - math.pi
print "Error is: " + str(Error)
```

Note: You must `include math` at the top of your program. You will need to replace `ApproxPi` with the name of your variable that stores the approximation of $\pi$.

`Error` will hold the error of your approximation. However, when we talk about the accuracy of an approximation, we often talk about how many digits the approximation is good to. For example, in one run of my `pi.py` program, I approximated $\pi$ to be `ApproxPi = 3.1438`. The first few digits of $\pi$ are 3.14159 so the error in my approximation was `Error = 0.00220734`. The first 2 digits after the decimal point of my approximation of $\pi$ are the same as those of $\pi$, so we say that my approximation is good to two digits. Notice that the first 2 digits after the decimal in `Error` are zero. The goal now is to make our Python program identify and report to us how many digits of accuracy you have in your approximation. Here is how my program looks when its run:

```
How many points would you like to draw: 100000
100000 points drawn, 78620 of which were inside the unit circle.
ApproxPi: 3.1448
Error: 0.00320734641021
Number of Correct Digits: 2
```

We can identify the number of digits in the approximation that are correct by using base 10 logarithms. Base 10 logarithms are implemented in Python in the math library as `math.log10()`, so if you want compute $\log_{10}(0.023)$ you would import the math library (`import math`) then type `math.log10(0.023)`. Try it!

**Question 6**: Define a logarithm. Do so in complete sentences *and* using mathematical formulae. What is a base 10 logarithm? What are the domain and range of the function $\log_{10} x$? (Cite your sources!)

**Question 7**: For each of the following values of `Error`, count how many zeros there are after the decimal, rewrite `Error` in scientific notation, calculate $\log_{10}|\text{Error}|$, then truncate $\log_{10}|\text{Error}|$ to an integer value by discarding all numbers after the decimal point. I've done the first one as an example.

| Error | # of zeros after decimal | Error in Sci. Not. | $\log_{10}\|\text{Error}\|$ | truncated $\log_{10}\|\text{Error}\|$ |
|---|---|---|---|---|
| 0.003207 | 2 | $3.207 \times 10^{-3}$ | -2.4938 | -2 |
| 0.000887 | | | | |
| -0.021592 | | | | |
| -0.093592 | | | | |
| 0.000093 | | | | |
| -0.003192 | | | | |

**Question 8**: In question 7, why did I make you calculate $\log_{10}|\text{Error}|$ and not $\log_{10}\text{Error}$?

**Question 9**: Consider the table in question 7. State a relationship between the number of zeros after the decimal and the exponent when you represent `Error` in scientific notation? Explain why (justify, prove) this relationship exists.

**Question 10**: Consider the table in question 7. State a relationship between the truncated $\log_{10}|\text{Error}|$ and the number of zeros after the decimal.

**Question 11**: Using known properties of logs, prove that $\log_{10}(m \times 10^{-a}) = (\log_{10} m) - a$ . Assume that $0 < m < 10$ and $a > 0$. If you don't know any properties of logs, ask google.

**Question 12**: For $1 \leq m < 10$, what is the range of $\log_{10} m$? In other words, fill in blanks: if $1 \leq m < 10$ then _____$< \log_{10} m <$ _____.

**Question 13**: Use your answers to questions 11 and 12 to justify (prove) your answer to question 10.

**Question 14**: How can you calculate (in Python) the number of accurate digits in your approximation of $\pi$?

## 6.1 Program Stage 4

Building on your program for Stage 3, complete your `pi.py` program. For this stage, add to the end of your Stage 3 program code that

1. Calculates the error in your approximation of $\pi$
2. Calculates the number of digits that are accurate in your approximation of $\pi$.

There is an example on the previous page of how my completed program behaves. Yours should behave similarly.

**Question 15**: Here is an example of (partial) output from my program:

```
ApproxPi: 3.13924
Error: -0.00235265358979
Number of Correct Digits: 2
```

Notice that my approximation of $\pi$ only matches $\pi \simeq 3.14159$ at the first digit after the decimal point, yet my program says it is correct to two digits. Here is another example:

```
ApproxPi: 3.14248
Error: 0.000887346410207
Number of Correct Digits: 3
```

Here my answer matches $\pi$ in the first 2 digits, but my program claims they match in the first 3. Explain what is happening in these examples. Can you justify my program's claim that the first two digits are correct in the first example and that first 3 digits are correct in the second example? If you can't justify it, can you fix it in your own program?

**Question 16**: How large do you need to make $n$ to get an approximation of $\pi$ that is accurate to 4 digits? When you run your program with this value of $n$, does it always produce an approximation of $\pi$ that is accurate to 4 digits? You should run your program at least 5 times with this value for $n$ and comment on the results.