Gold Problem 2: Pi from pie

Copied from: https://www.cs.hmc.edu/twiki/bin/view/CS5/PiFromPieGold on 3/22/2017

[30 points; individual or pair]

Filename: `hw8pr2.py`

It is perhaps surprising that it is possible to estimate the mathematical constant π without resorting to any techniques or operations more sophisticated than counting, adding, and multiplication. This problem asks you to write two functions that estimate pi (`3.14159...`) by *dart-throwing*.

Computing Pi from Pie: background

Imagine a circle inscribed within a square that spans the area where `-1 ≤ x ≤ 1` and `-1 ≤ y ≤ 1`. The area of the inscribed circle, whose radius is `1.0` would be π .

If you were to throw darts at random locations in the square, only some of them would hit the circle inscribed within it. The ratio

```
area of the circle / area of the square
```

can be estimated by the ratio

```
number of darts that hit the circle / total number of darts thrown
```

As the number of darts increases, the second ratio, above, gets closer and closer to the first ratio. Since three of the four quantities involved are known, they can be used to approximate the area of the circle - this in turn can be used to approximate π

Designing your dart-throwing...

To throw a dart, you will want to generate random x and y coordinates between `-1.0` and `1.0`. Be sure to include the line

```
import random
```

near the top of your file. When you do this, you will now be able to use the function
`random.uniform( -1.0, 1.0 )`
That line will return a floating-point value that is in the range from `-1.0` to `1.0` . For example, you will be able to write
`x = random.uniform( -1.0, 1.0 )`

Helper function to write: throwDart()

With this background in mind, many have found it helpful to write a helper function that

- throws one "dart" at the square by generating getting a random `x` and a random `y` coordinate between -1 and 1
- determines whether that dart is within the circle of radius 1 centered at the origin -- you can use the `math.sqrt` function to check this, though you may note that it's not strictly necessary!
- returns `True` if the dart hits the circle and `False` if the dart misses the circle
- remember that the dart will *always* hit the square, by the way the throw is designed...

This helper function could be used for both of this problem's main functions: `forPi` and `whilePi`.

However you design your Monte Carlo simulation, you should be sure - as always - to include an explanatory docstring for each of your functions!

Main function to write #1: forPi(n)

Your `forPi( n )` function will take in a positive integer `n` as input.

It should "throw" `n` darts at the square.

Each time a dart is thrown, the function should print

- the number of darts thrown so far
- the number of darts thrown so far that have ***hit*** the circle
- the resulting estimate of π

Return value - *be sure to do this!*

The `forPi` function should return the *final resulting estimate of π* after `n` throws.

Here is an example run to show how `forPi` should work:

- Your printing will vary because of the randomness... .
- However, it should converge to the real value of π as the number of darts, `n` gets larger

```
In [1]: forPi( 10 )
1 hits out of 1 throws so that pi is 4.0
2 hits out of 2 throws so that pi is 4.0
3 hits out of 3 throws so that pi is 4.0
4 hits out of 4 throws so that pi is 4.0
4 hits out of 5 throws so that pi is 3.2
5 hits out of 6 throws so that pi is 3.33333333333
6 hits out of 7 throws so that pi is 3.42857142857
6 hits out of 8 throws so that pi is 3.0
7 hits out of 9 throws so that pi is 3.11111111111
8 hits out of 10 throws so that pi is 3.2

Out[1]: 3.2
```

Main function to write #2: whilePi(error)

Your `whilePi( error )` function will take as input a positive floating-point value, `error`.

It should then proceed to throw darts at the dartboard (the square) until the *absolute difference* between the function's estimate of π and the real value of π is less than `error`.

Your `whilePi` function requires the actual, known value of π in order to determine whether or not its estimate is within the error range! Although this would not be available for estimating a truly unknown constant, for this function you include the line

```
import math
```

in your code and then use the value of `math.pi` as the actual value of π .

Similar to your `forPi` function, for each dart throw your `whilePi` function should print

- the number of darts thrown so far
- the number of darts thrown so far that have ***hit*** the circle
- the resulting estimate of π

after each dart throw it makes.

Return value - *be sure to do this!*

The `whilePi` function should return the *number of darts thrown* in order to reach the input accuracy.

Here is an example run to show how `whilePi` works:

```
In [7]: whilePi( 0.1 )
1 hits out of 1 throws so that pi is 4.0
2 hits out of 2 throws so that pi is 4.0
3 hits out of 3 throws so that pi is 4.0
4 hits out of 4 throws so that pi is 4.0
5 hits out of 5 throws so that pi is 4.0
5 hits out of 6 throws so that pi is 3.33333333333
6 hits out of 7 throws so that pi is 3.42857142857
7 hits out of 8 throws so that pi is 3.5
7 hits out of 9 throws so that pi is 3.11111111111

Out[7]: 9
```

Submission

Be sure to submit your `hw8pr2.py` file in the usual way...