Problem 2: Pi from Pie

Favoris

Week 8: Loops > Homework 8 > Problem 2: Pi from Pie

It is surprising that it's possible to estimate the irrational mathematical constant π arbitrarily well without resorting to any techniques or operations more sophisticated than counting, adding, and multiplication.

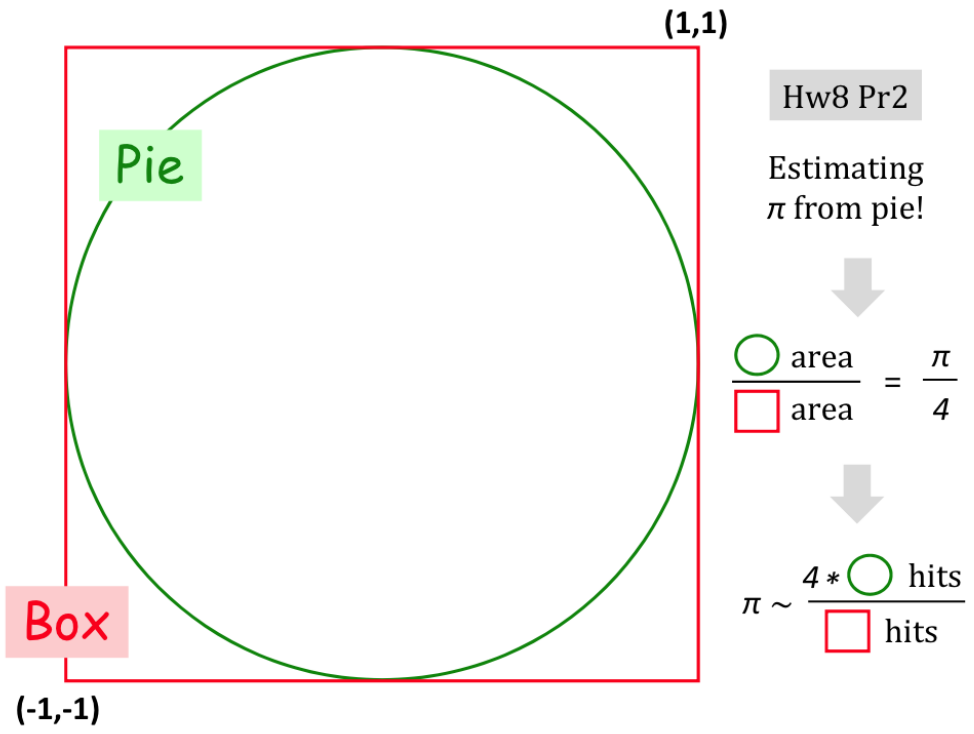
This problem asks you to write two functions that estimate pi (`3.14159...`) by *dart-throwing*.

Get started by creating a new trinket for this problem.

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.

Here is a graphical overview:



And a text description:

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 circlearea of the square
can be estimated by the ratio
number of darts that hit the circletotal 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—and this in turn can be used to approximate π!

Designing your dart-throwing

To throw a dart, first create a function named `dartThrow()`, which should:

- First, generate random `x` and `y` coordinates between `-1.0` and `1.0`. Give the coordinates reasonable names! For example, `x = random.uniform(-1.0, 1.0)`. Please do use `random.uniform(-1.0, 1.0)` to generate these values. Otherwise, our grader won't be able to grade your homework correctly!
- Be sure to include the line `import random` in your file!
- Then, your function should **check** if the random (x,y) point is **within a circle of radius one**. (You already know it's within the square with bounds from -1 to 1.)
- Your function should return `True` if the dart hit the target (it's in the circle).
- Your function should return `False` if the dart did not hit the target (it's not in the circle).

How to check if it's in the circle?!

- The distance of the dart to the origin is x2+y2
- You can use `math.sqrt` or `something**0.5` in order to take square roots
- Then, if the distance of the dart to the origin is less than `1.0`, it's ***in*** the circle.
- if the distance of the dart to the origin is greater than than `1.0`, it's ***not in*** the circle.
    - You won't need to worry about less-than vs. less-than-or-equal (whether it hits *on* the circle).
    - In fact, you don't really need the square root, either!

Try your function out! Call `dartThrow()` several times.

- Although it's random, be sure that it does not **always** return one of `True` or `False`.
- In fact, it should return `False` about, but not exactly, 1/4 of the time. (Why?!)

Pi from pie function #1: `forPi( n )`

Next, write a function `forPi( n )` function will take in a positive integer `n` as input and

- It should "throw" `n` darts at the square using your `throwDart()` function.
- Since you know the number of throws, a `for` loop is best!

- A loop that will run `n` times is `for i in range(n):`
- Be sure to start a couple of useful variables with reasonable initial values (`0`), perhaps `numthrows` and `numhits`.

Within the `for` loop, each time a dart is thrown, the function should **print** (not return):

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

Remember that you need to use `4.0` times the number of hits over the number of throws—the floating-point value is important! See below for an example.

Return value

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

Here is an example run. Because of the randomness, exact results will vary (but should be reasonable)!

```
>>> 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
3.2
```

Pi from pie function to write #2: `whilePi( maxerror )`

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

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 `maxerror`.

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, we can use Python's built-in value of π. You should include the line `import math` in your code and then use the value of `math.pi` as the actual value of π.

Just as `forPi` used a `for` loop, you will want `whilePi` to use a `while` loop:

- You'll need several variables to be initialized before the `while` loop. I used `numthrows`, `numhits`, and `estpi`, my name for my estimate of π.
- The `while` loop will need to compare the value of `abs(estpi—math.pi)` with the value of `maxerror`.
- **Don't call `forPi` from within `whilePi`!** It's better to copy parts of the `forPi` code for reuse, instead.
- Similar to your `forPi` function, after 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 (I called it `estpi`)

Return value

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

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

```
>>> 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
9
```

Submit Homework 8, Problem 2

25.0/25.0 points (graded)

To submit your Homework 8, Problem 2 code, you'll need to copy it from the your trinket or file and paste it into the box below. After you've pasted your code below, click the "Check" button.

IMPORTANT: Make sure that there aren't spaces at the beginning of your code, and that you copied all of the characters. If there are extra spaces or you are missing spaces, our server won't be able to run your code and we won't be able to give you any of the points you deserve for your hard work.

1

2

```
import random
import math

def dartThrow():
  """Generate random x and y coordinates between -1.0 and
1.0
```

```
  return True if the dart hit the target
  return False if the dart did not hit the target
"""
  x = random.uniform(-1.0, 1.0)
  y = random.uniform(-1.0, 1.0)
  if (x ** 2 + y ** 2) ** 0.5 <= 1.0:
    return True
else:
    return False
```

Press ESC then TAB or click outside of the code editor to exit
correct

correct

Test results
CORRECT See full outputSee full output

You have used 1 of 3 attempts Some problems have options such as save, reset, hints, or show answer. These options follow the Submit button.