CS1210 Lecture 26 Oct. 22, 2021

- HW6 due Thursday
  - It's certainly fine to import the random module for HW6 (for the Nim problem)
- Quiz 3 next Friday, Oct. 29
- Last time:
  - Finished object oriented programming (Ch 17, 18, 19)
- Today: Some problems involving randomization and simulation

Using randomization to mix up (shuffle) a list of numbers

```
def mixup(L):
    newL = L[:]
    length = len(L)
    for i in range(length):
        newIndex = random.randint(0,  length-1)
        newL[newIndex], newL[i] = newL[i], newL[newIndex]
    return(newL)
```

- What do you think?
- Test on a few lists. mixupTests.py

  >>> mixup([1,2,3,4,5])

  [1, 3, 4, 5, 2]
- Run testMixup(100000). What do you expect as result?
- Hmm…
- See: http://blog.codinghorror.com/the-danger-of-naivete/ and https://www.datamation.com/entdev/article.php/616221/How-We-Learned-to-Cheat-at-Online-Poker-A-Study-in-Software-Security.htm

A little Monte Carlo simulation

Roughly speaking, Monte Carlo simulation is a fancy name for using repeated random sampling of a problem space to determine results

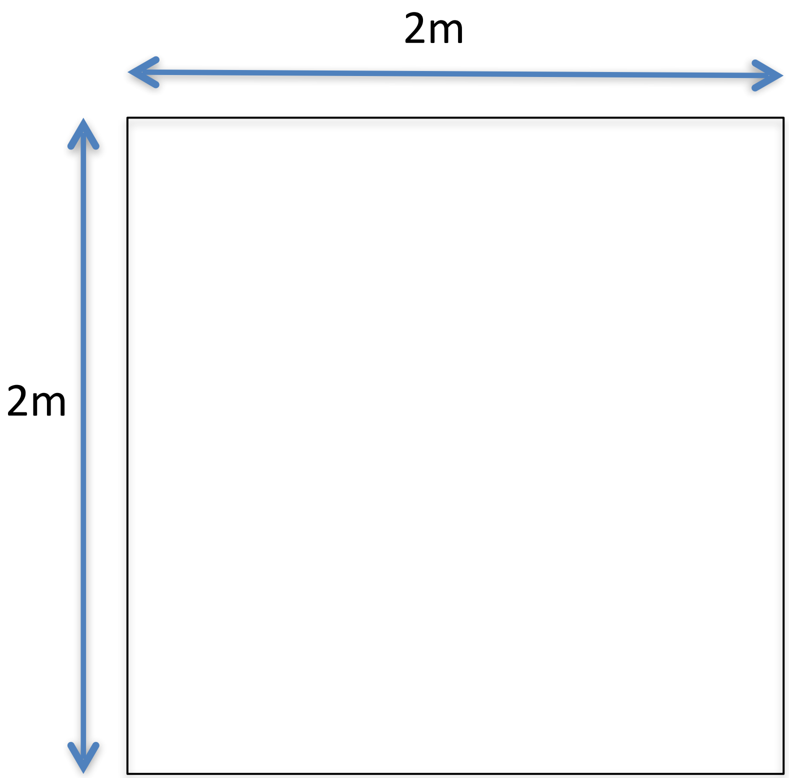
lec26coins.py is a very simple Monte Carlo simulation of coin flipping.

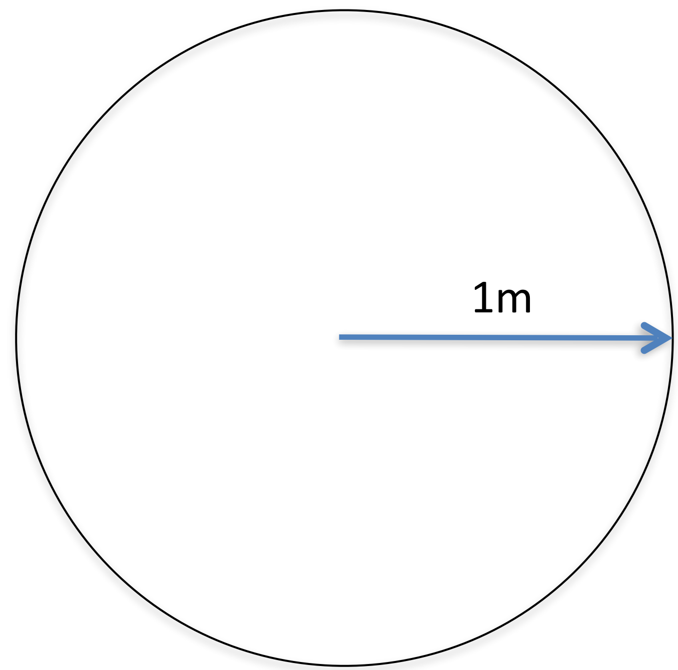
- doNCoinFlips(numFlips): return number of heads and number of tails resulting from specified number of flips
- doCoinFlipTrials(numTrials, numCoins): doing *one* set of flips is not usually a good way to do an experiment. Better to do several "trials" flipping the same number of coins. E.g flip 100 coins 10 times

  Compute, print, and return statistical info:
  - average head/tail ratio (averaged over the set of trials)
  - std deviation of head/tail ratios

- doCoinFlipExperiment(minCoins, maxCoins, factor)
  - collect doCoinFlipTrials data for different numbers of coins, so we can see trend of statistics as number of flips grows:
    - minNum, minNum*factor, minNum*factor*factor, …
- plotResults
  - use pylab to create graphs of stats gathered by doCoinFlipExperiment

How can we use Monte Carlo simulation to calculate the value of π?

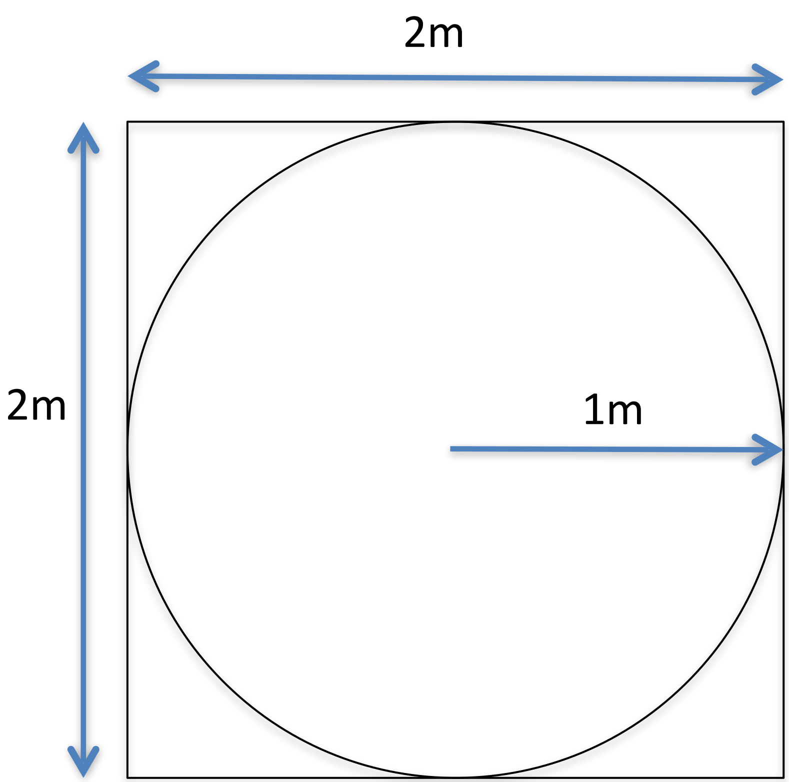




Area? 4 sq m

Area? π sq m

How can we use Monte Carlo simulation to calculate the value of π?



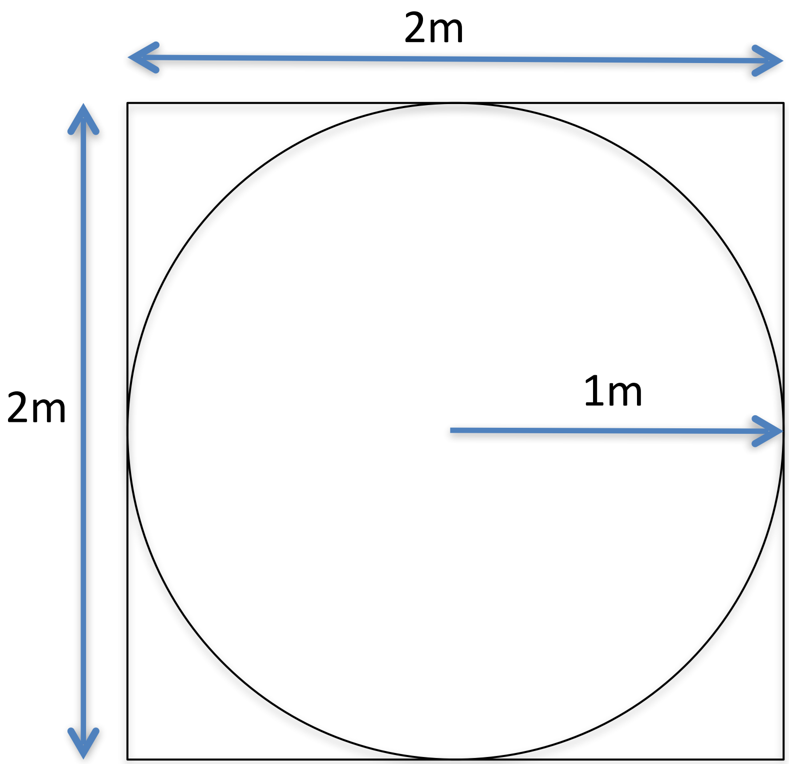
Square area: 4 sq m
Circle area: π sq m

Ratio of circle area to square area? π/4

If we drop a million grains of sand in the square, what fraction of them will be in the circle? π/4

How can we simulate dropping that sand?

Monte Carlo simulation to calculate π



Square area: 4 Circle area: π

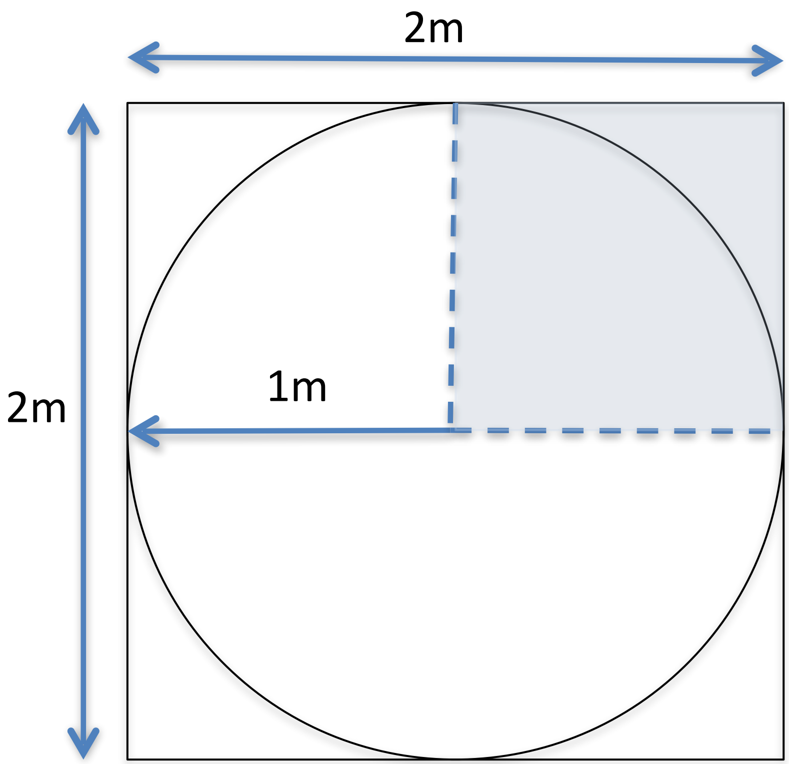
If drop 1000000 grains of sand in square, fraction π/4 should fall in circle

How can we simulate dropping that sand?

1. Generate 1000000 2d coordinates (x,y) with x and y both random between in range [-1, 1]
2. Count fraction f that are in circle!

Finally, calculate π as:
f * 4

Monte Carlo simulation to calculate π



Square area: 4   Circle area: π
Quadrant area: 1
Quarter circle area: π/4
Quarter circle/quadrant ratio: π/4, the same as before

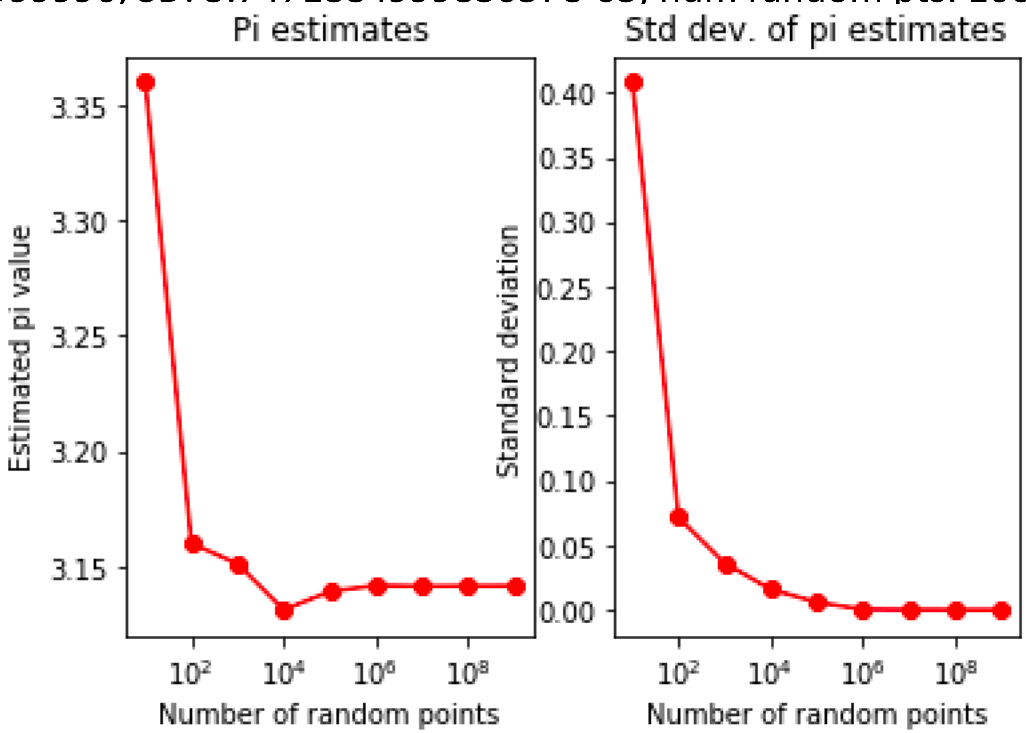
lec26pi.py (estimatePi function) implements this simulation, with one small modification. Drop the grains of sand only in the upper right quadrant.

1. Generate the 1000000 2d coordinates (x,y) with x and y in range [0, 1]
2. Count fraction f that are in circle! How?
   *in circle if x*x + y*y <= 1*

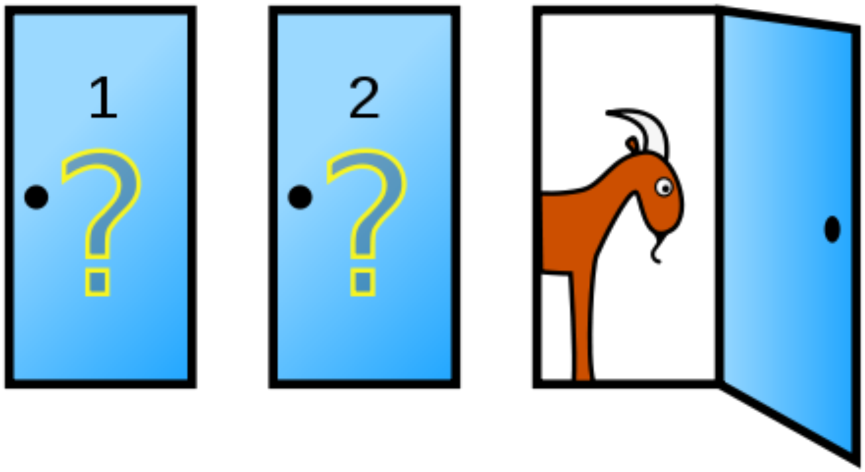
Finally, calculate π as:
f * 4
(Note: although a good example, this is not an efficient way to calculate pi)

Read more at: https://learntofish.wordpress.com/2010/10/13/calculating-pi-with-the-monte-carlo-method/

```
>>> r = findPi(10, 1000000000, 10)
Estimate: 3.3600000000000003, SD: 0.40792156108742283, num random pts: 10
Estimate: 3.16, SD: 0.07155417527999319, num random pts: 100
Estimate: 3.1512000000000002, SD: 0.03616849457746344, num random pts: 1000
Estimate: 3.1312, SD: 0.0158634170341702, num random pts: 10000
Estimate: 3.139152, SD: 0.005902536403953735, num random pts: 100000
Estimate: 3.1417592, SD: 0.0003406290651133859, num random pts: 1000000
Estimate: 3.1414109600000004, SD: 0.00019791773644632456, num random pts: 10000000
Estimate: 3.141578496, SD: 0.00013422946466409779, num random pts: 100000000
Estimate: 3.1415904263999996, SD: 5.747188499983057e-05, num random pts: 1000000000
```



Monty Hall problem http://en.wikipedia.org/wiki/Monty_Hall_problem



Suppose you're on a game show, and you're given the choice of three doors: Behind one door is a car; behind the others, goats. You pick a door, say No. 1, and the host, who knows what's behind the doors, opens another door, say No. 3, which has a goat. He then says to you, "Do you want to pick door No. 2?"

The problem: Is it to your advantage to switch your choice?

- Parade magazine, 1990, Marilyn vos Savant’s “Ask Marilyn column”

Monty Hall problem

- M. vos Savant (whose column generated huge public awareness) quotes psychologist M. Piattelli-Palmarini: "... no other statistical puzzle comes so close to fooling all the people all the time." Herbranson and Schroeder: Pigeons repeatedly exposed to the problem show that they rapidly learn always to do the right thing, unlike humans. ☺
- Demonstrate simulation using randomization to help "see" that switching is the right thing to do. montyhall.py

An problem to think about for fun

You want to sell your phone.

A large line of people assembles seeking to buy it. Each has a random price offer (bid) in mind. You want to maximize price BUT you must consider the bids ONE AT A TIME, in the order received, and REJECT OR ACCEPT EACH ONE IMMEDIATELY.

Is there are strategy that can make it likely you get a good price?

E.g. "always take first bid" – chance of getting best price is then 1/n. Not so good….

Monday/Wednesday

- Algorithm run-time analysis and computational complexity
- Introduce searching and sorting