CS1 Lecture 36 Apr. 17, 2017

- HW 8 due 9am Wed.
- Exam 2: April 20, 6:30-8:00pm, C20 Pomerantz Center
  - Main topics: dictionaries (Ch 11), classes and objects (Ch 15-18), algorithm analysis (Ch 21), sorting and searching, graphs.
  - Full review on Wed., with review document and sample exam questions
- Discussion sections this week: intro practice with GUIs. Important for HW9 and HW10 (but GUIs will not be on Exam 2)

Last time

- Some problems involving randomization and simulation
  - Monty Hall
  - Coin flipping
  - Mixup

Today

- One more problem solution using randomization
- Introduction to GUI programming with Python

HW8 Q1

Said basic O(n^2) method for building graph is okay … Consider some variants. E.g.:

```
for word1 in wordList:
    node1 = wordGraph.getNode(word1)
    for word2 in wordList:
        node2 = wordGraph.getNode(word2)
        if shouldHaveEdge(…) :
            if not wordGraph.hasEdge(…):
                wordGraph.addEdge(…)
```

HW8 Q1

Said basic O(n^2) method for building graph is okay … Consider some variants. E.g.:

```
for word1 in wordList:
    node1 = wordGraph.getNode(word1)
    for word2 in wordList:
        node2 = wordGraph.getNode(word2)
        if shouldHaveEdge(…) :
            if not wordGraph.hasEdge(…):
                wordGraph.addEdge(…)
```

This is O(n^3), due to O(n) getNode in inner loop!
And for words5.txt with ~2500 words, 2500x2500x2500 is too slow!

Last time - 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

lec35coins.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

Last time - HW8 Q2

Suppose we flip 15 coins and the result is

H, T, T, H, H, H, T, H, H, T, H, T, T, T, T

The length of the longest consecutive "run" of heads is 3.

If you flip 100 coins, what do expect will be the length of the longest consecutive string of heads?

How can we figure this out?

Flip 100 coins once? No reason to believe result is close to expected value if we do it again

Better: flip 100 coins several times and take average.

How about 200 coins? 1000 coins?

As number of coins grows, how do expect long run length grow?

HW8 Q2: modify lec35coins.py

to try to answer this problem

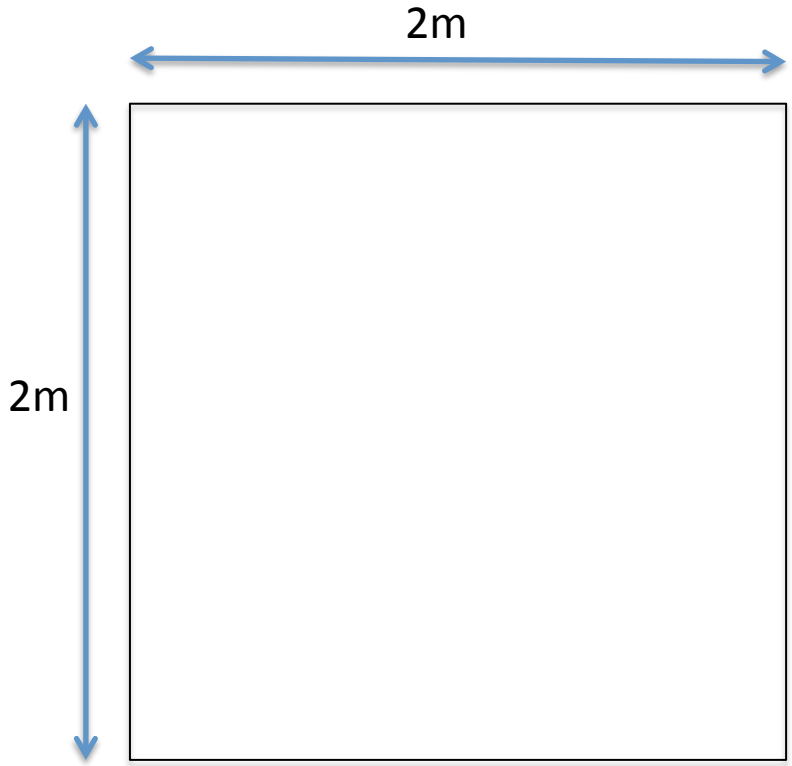
HW8 Q2

- First, modify doNCoinFlips function to return (instead of the numbers of heads and tails) the length of the longest run of consecutive heads found in a sequence of n flips.

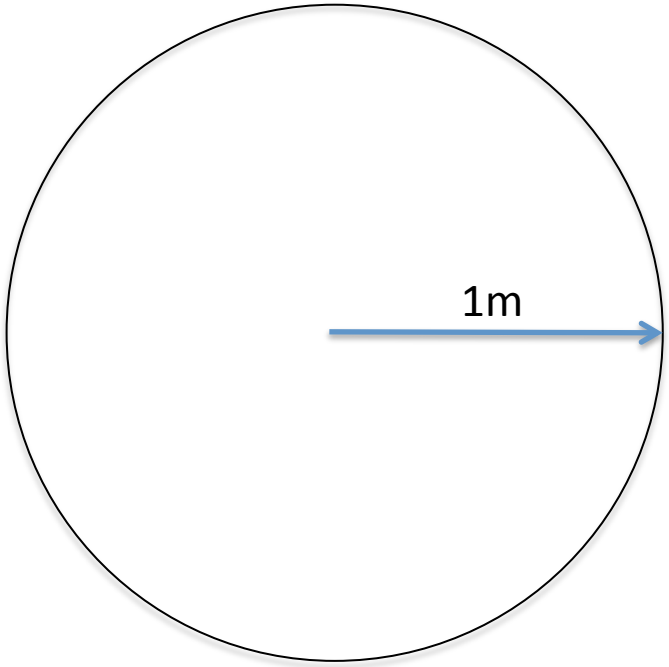
- Next, modify doCoinFlipTrials to collect and process the results of multiple trials of coin flipping, calculating the average longest run of heads for the set of trials.

- Third, modify doCoinFlipExperiment to calculate and print the expected longest heads run length for a wide range of numbers of flipped coins.

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

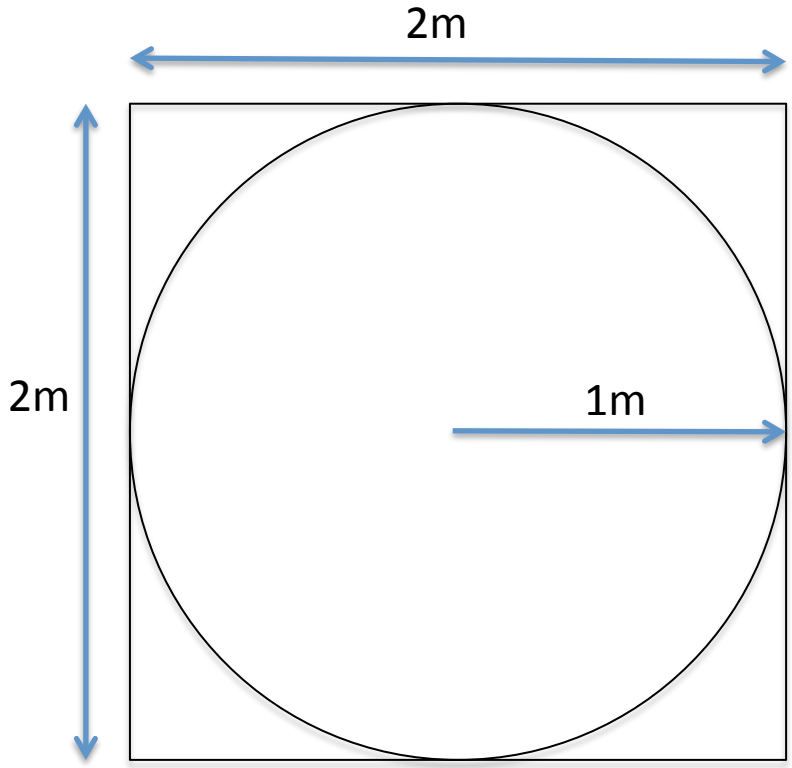


Area? 4 sq m



Area? π sq m

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



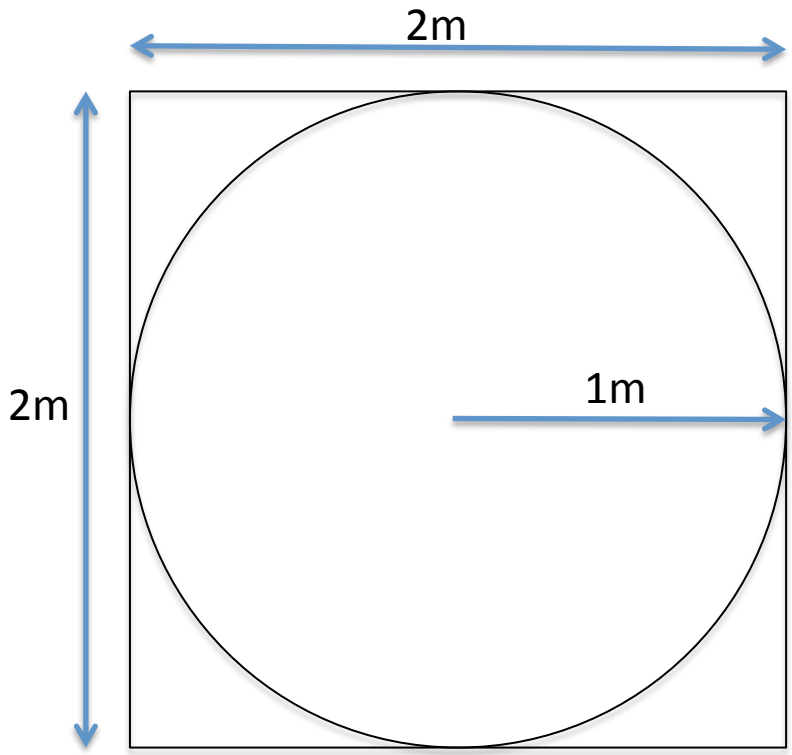
Square area: 4 sq m
Circle area: π sq m

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

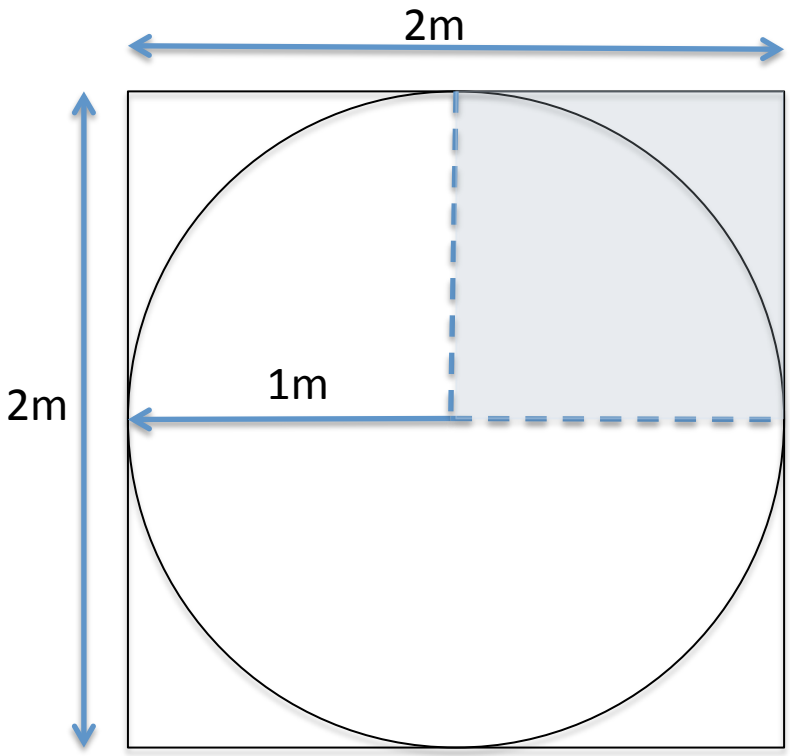
If drop 1000000 grains of sand in square, fraction π/4 should fall in circle

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

Lec36pi.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)

Next topic: Graphical User Interface(GUI) Programming in Python

- tkinter is a commonly used Python layer on top of a standard GUI toolkit TCL/Tk
- GUIs built using *widgets:* basic building blocks including buttons, labels, text, entry fields, scroll bars, etc.
- Steps:
  - Define widgets and layout
  - Specify how to handle “events” (button presses, mouse clicks, etc.)
  - Start GUI/”event loop”

Getting started with tkinter

1. import tkinter
   - ***NOTE***: module was called Tkinter in Python 2. Much web documentation still calls is Tkinter. Functions, etc.. are the same but make sure to use 'tkinter' not 'Tkinter' when important module, calling functions, etc.
2. Before using any other tkinter functions, you must call Tk() to create root/main window and initialize Tkinter
   - e.g myMainWindow = Tk()
3. Next, define other GUI widgets – buttons, labels, entries, and their placement (via pack/grid function calls)
4. Specify how to handle event (e.g. button presses), typically by defining "callback functions" that are called by the GUI system when it detects events
5. Finally, when ready to start execution of the GUI, call mainloop()
   - e.g. myMainWindow.mainloop()

Note: mainloop() gives control of execution to Python. It won't return direct control to you until after you close/kill the main window. So, no commands/ function calls should follow it in your code. After you call it, you only get an opportunity to change things when *events* occur – i.e. when you click the mouse on buttons, type text in entries, etc.

- tkinter is big – there are long chapters and even whole books about it. You will use Tkinter in HW9 and HW 10 but not a lot of the features.
- You just need to understand the basics of a few widget types (Label, Entry, Button, Frame) and how to respond to events like button presses.
- Make sure you understand these four small examples (links available with Lec 36 on class web page)
  - minimal.py
  - minimal2.py
  - simplegui1.py
  - simplegui2.py

  and the example that you'll build upon in discussion section tomorrow

Links/resources for learning tkinter

- This tutorial - http://www.tutorialspoint.com/python/python_gui_programming.htm - does a good job of explaining and demonstrating the basics and includes several good small examples. It seems like a good place to start.
- If you want a "real textbook chapter", Chapter 6 of Kent Lee's book "Python Programming Fundamentals" is pretty good. The book is electronically available to UI students through the UI library (you need to be on the campus network to access it). http://link.springer.com/chapter/10.1007%2F978-1-84996-537-8_6
- This site can also very helpful. It's usually the first hit when I do Google search (though I'm not sure all info there is fully up-to-date): http://effbot.org/tkinterbook/tkinter-index.htm
- Tkinter info on the official Python site:
- https://docs.python.org/3.5/library/tkinter.html
- Some of the explanations here can be helpful (e.g. explains "pack" better than many others): http://thinkingtkinter.sourceforge.net

Next time

- more GUI
- review for exam 2