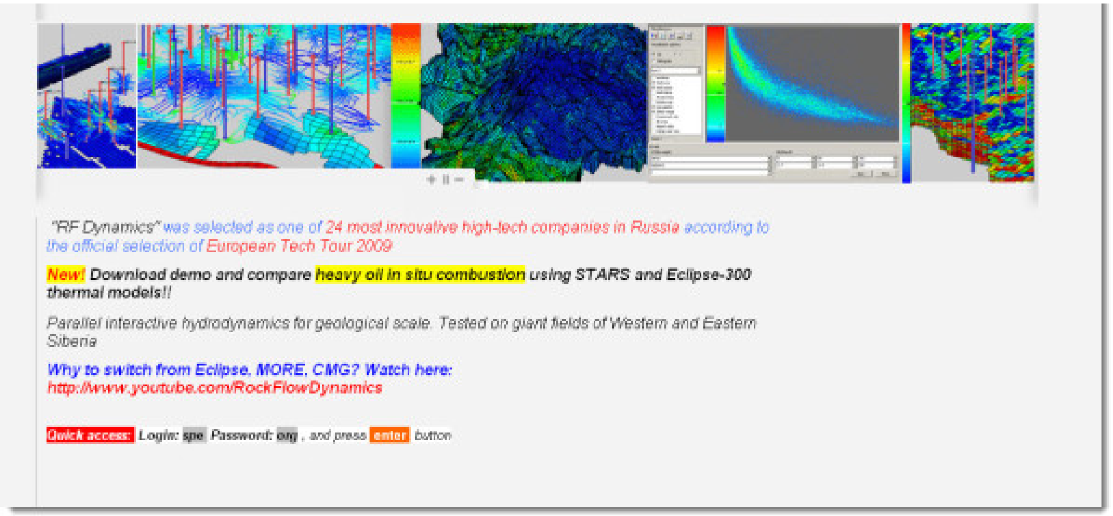
Monte Carlo Techniques

Professor Stephen Sekula

Guest Lecture – PHYS 4321/7305

What are “Monte Carlo Techniques”?

- Computational algorithms that rely on repeated random sampling in order to obtain numerical results
- Basically, you run a simulation over and over again to calculate the possible outcomes, either based on probability or to determine probability
- Like playing a casino game over and over again and recording all the game outcomes to determine the underlying rules of the game
- Monte Carlo is a city famous for its gambling – hence the name of this class of techniques



"RF Dynamics" was selected as one of 24 most innovative high-tech companies in Russia according to the official selection of European Tech Tour 2009

***New!* Download demo and compare heavy oil in situ combustion using STARS and Eclipse-300 thermal models!!**

Parallel interactive hydrodynamics for geological scale. Tested on giant fields of Western and Eastern Siberia

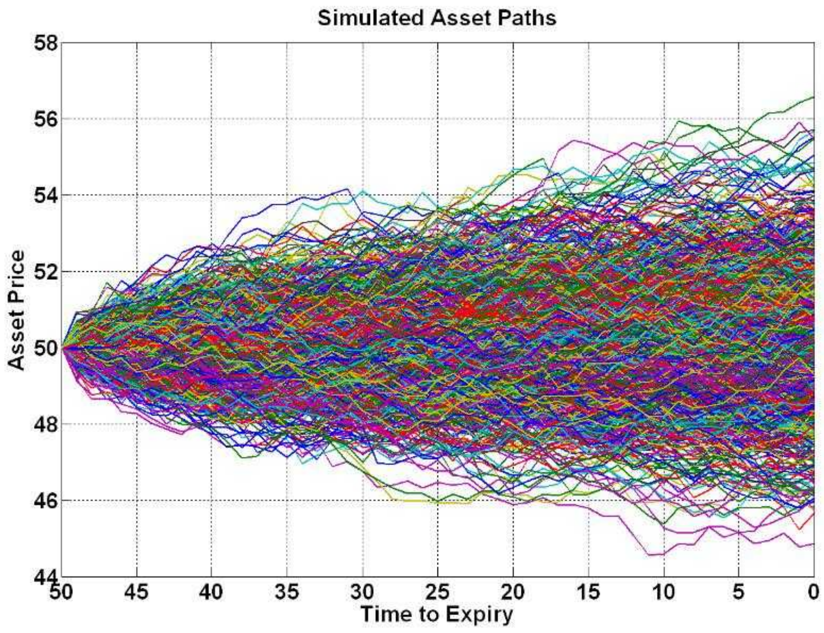
Why to switch from Eclipse, MORE, CMG? Watch here:
http://www.youtube.com/RockFlowDynamics

***Quick access:* Login: spe Password: org , and press enter button**

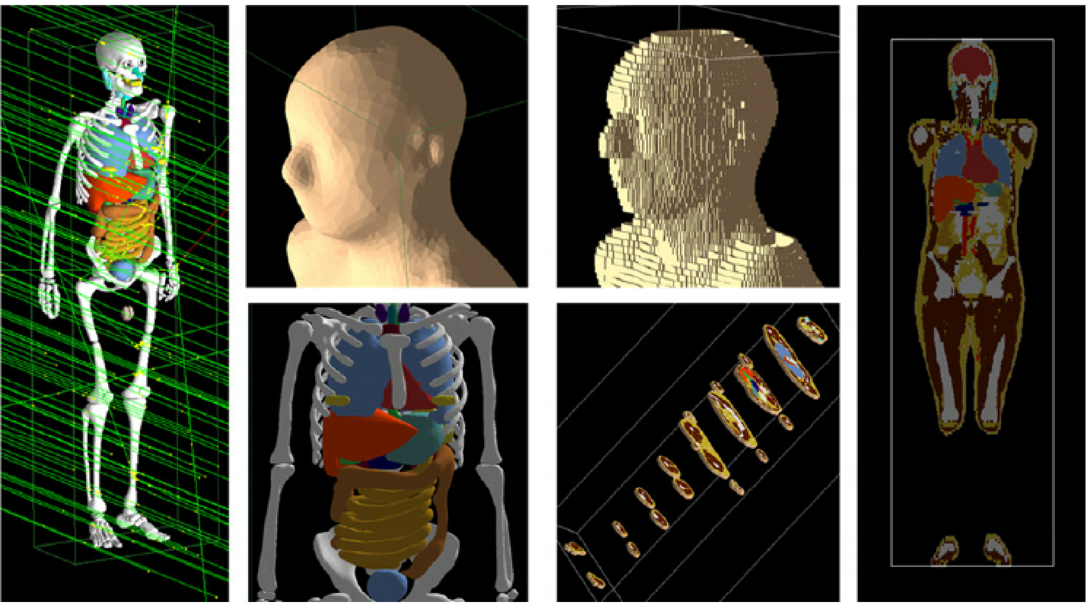
RFD
Rock Flow Dynamics

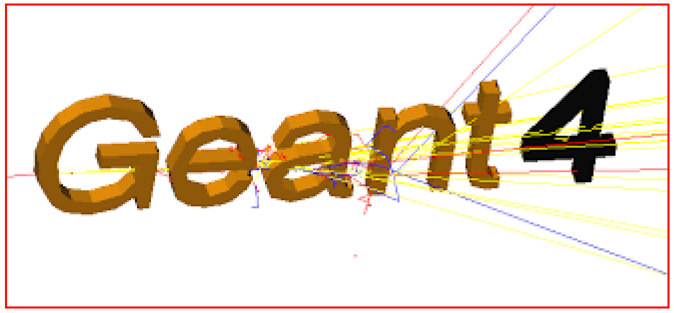
http://rfdyn.com/

Monte Carlo modeling of geological hydrodynamics for hydrocarbon extraction



MATLAB is used to Monte Carlo simulate financial evolution

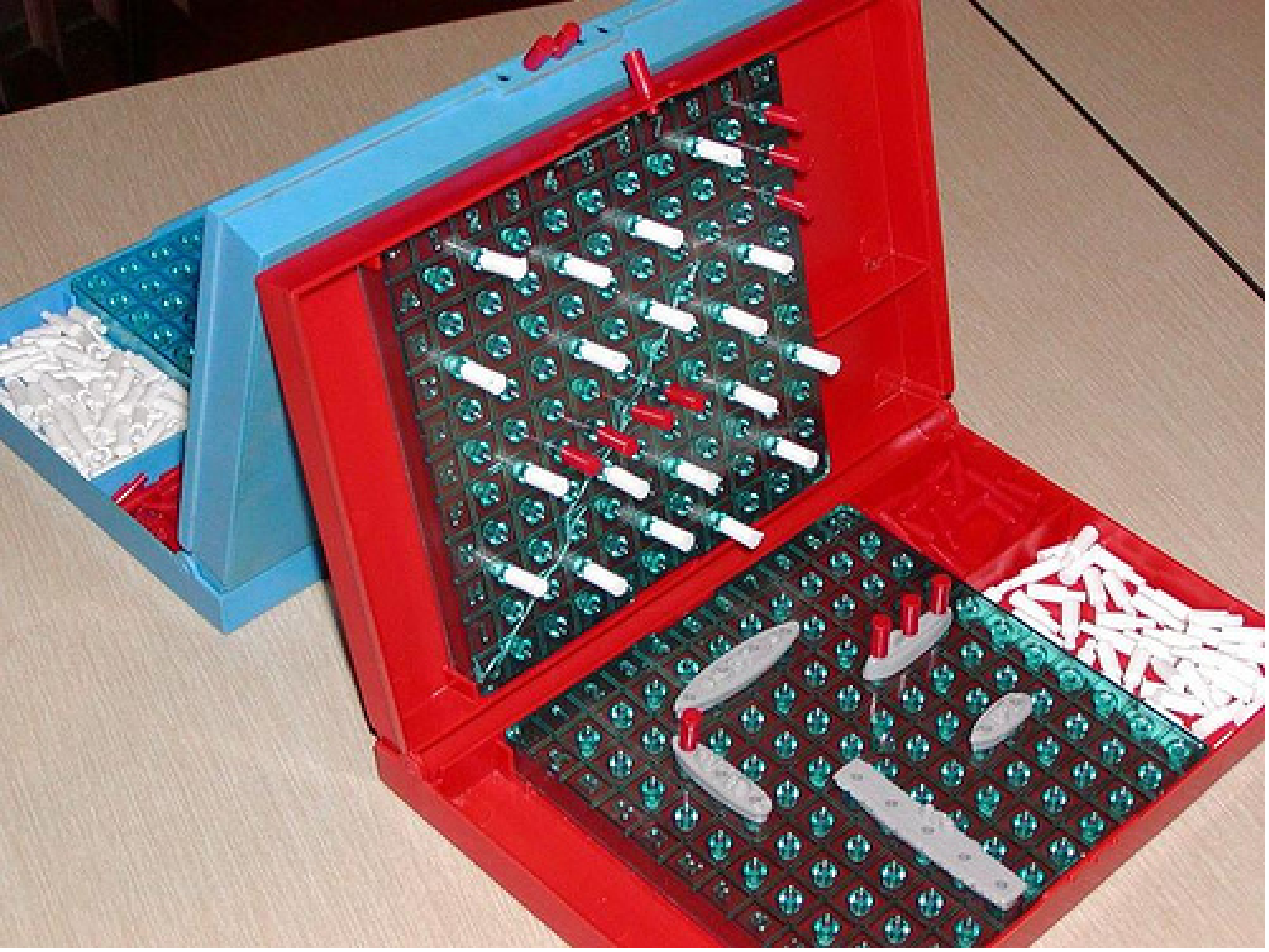




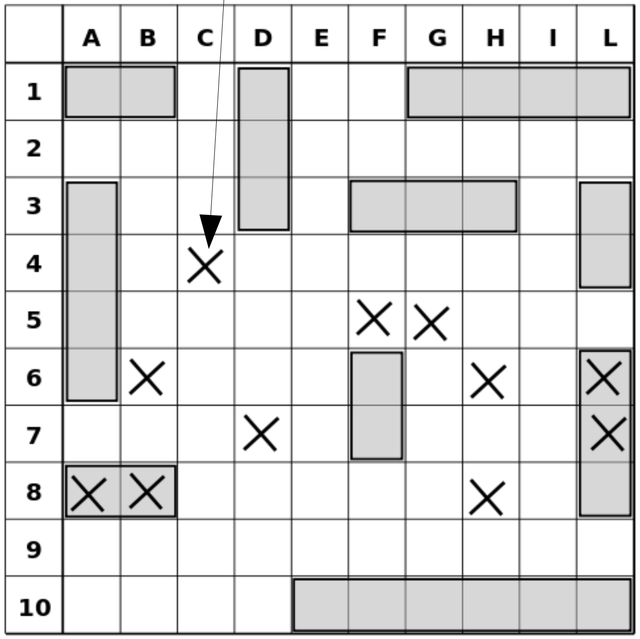
Design and use of “phantoms” to model radiation dose in human tissue – necessary for prepping a patient for radiation-based cancer therapy.

HAVE YOU EVER (KNOWINGLY) USED “MONTE CARLO TECHNIQUES”?

EVER PLAYED "BATTLESHIP"?



MISSES



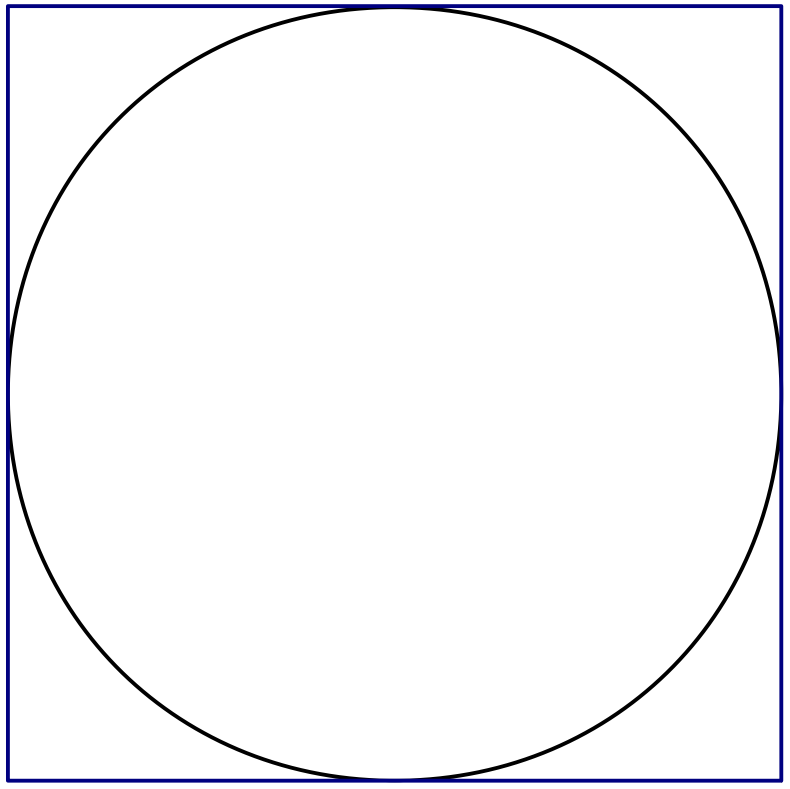
HITS

IF SO, YOU HAVE APPLIED MONTE CARLO TECHNIQUES.

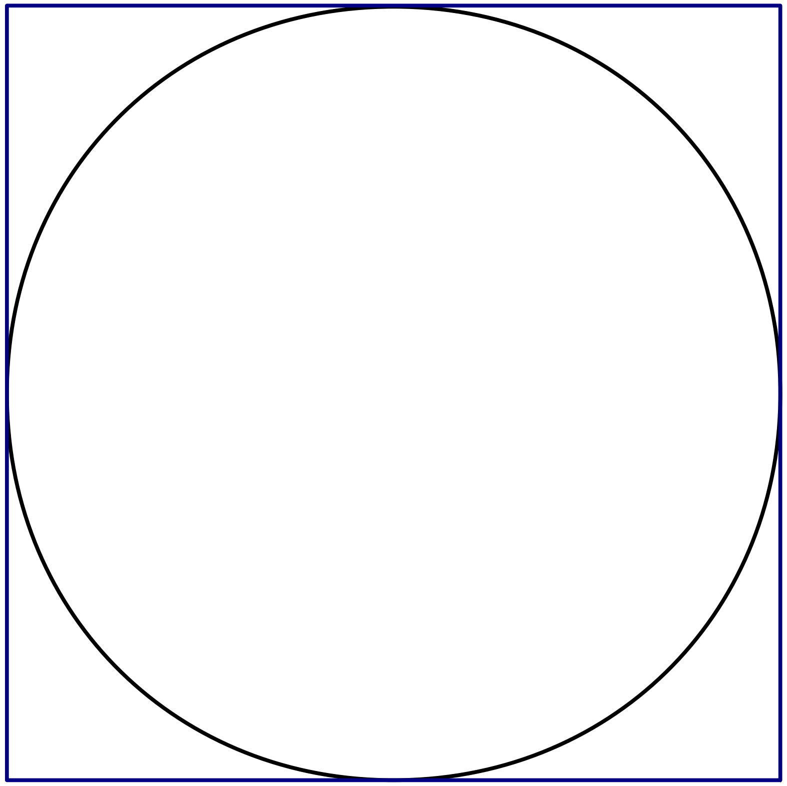
A Simple Physical Example

- Let's illustrate this class of techniques with a simple physical example: numerical computation of π
- π: the ratio of the circumference of a circle to its diameter: 3.14159265359...
- It's difficult to whip out a measuring tape or ruler and accurately measure the circumference of an arbitrary circle.
- The Monte Carlo method avoids this problem entirely

Begin by drawing a square, inscribed into which is a circle. The properties of the square are much easier to measure.

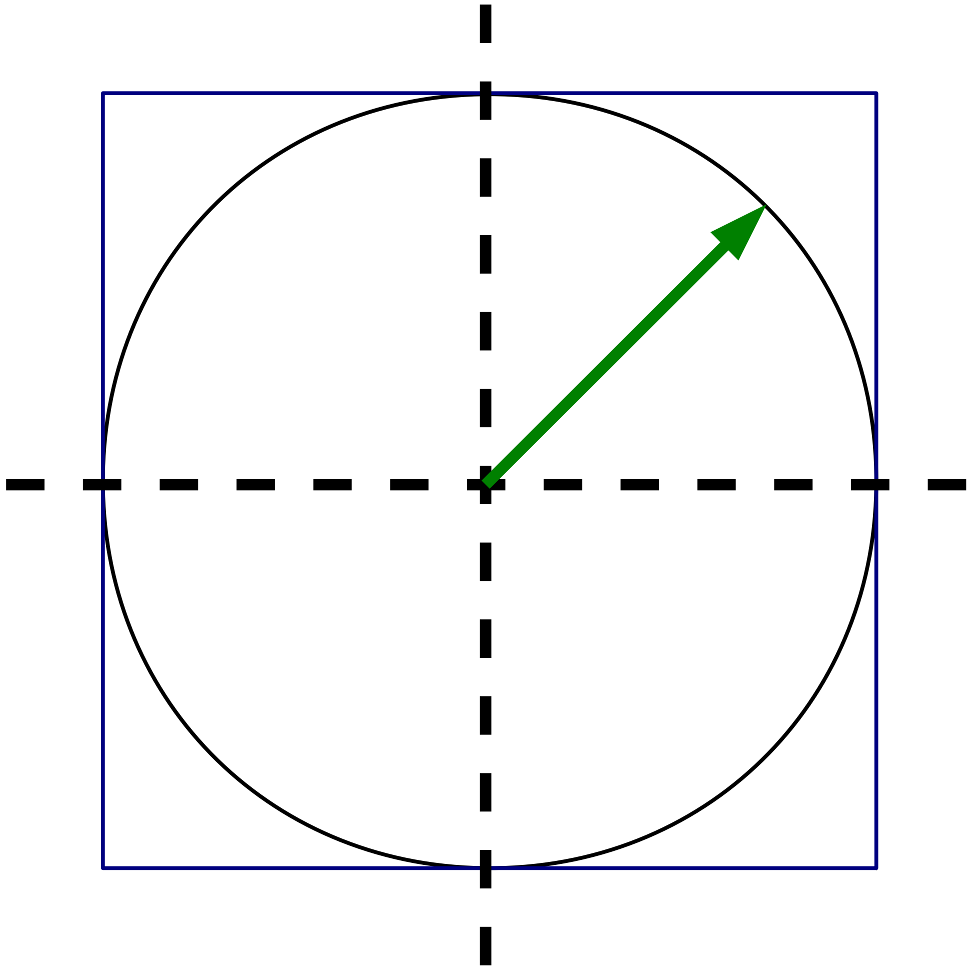


What do we know?

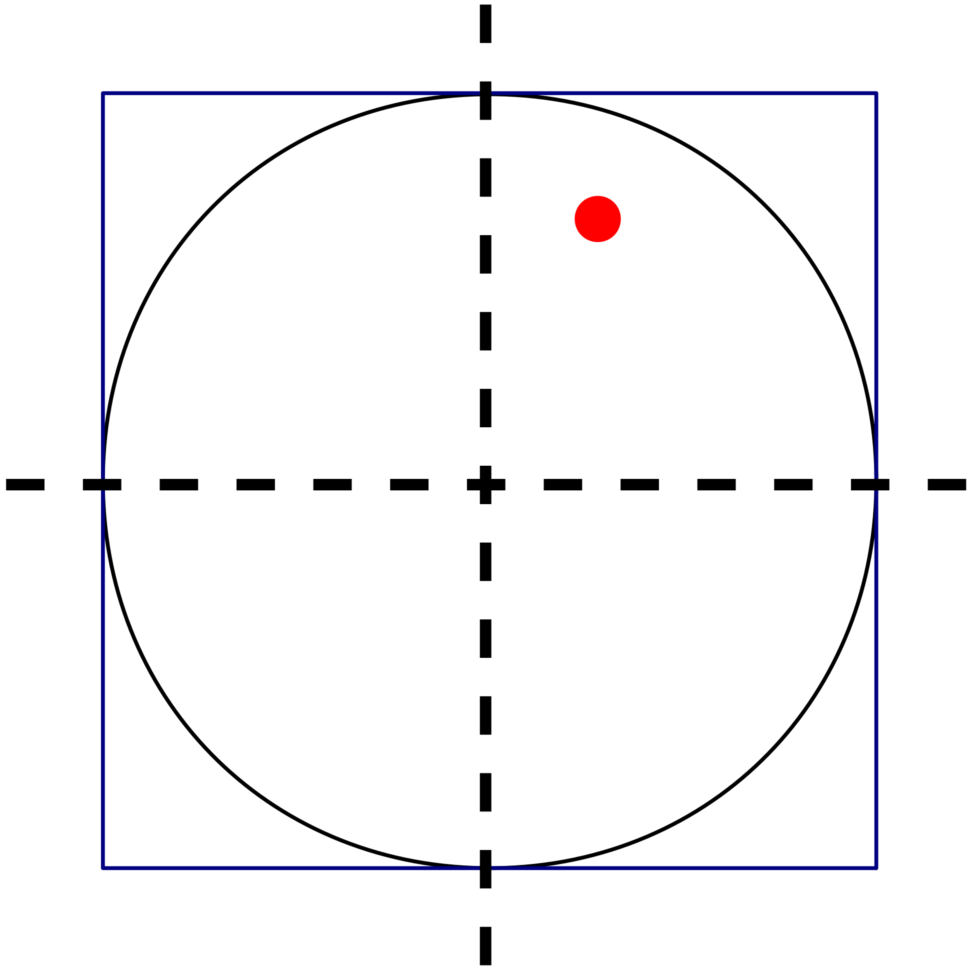


We know the relationship between the radius of a circle and the x and y coordinate of a point on the circle boundary:

$$r = \sqrt{x^2 + y^2}$$



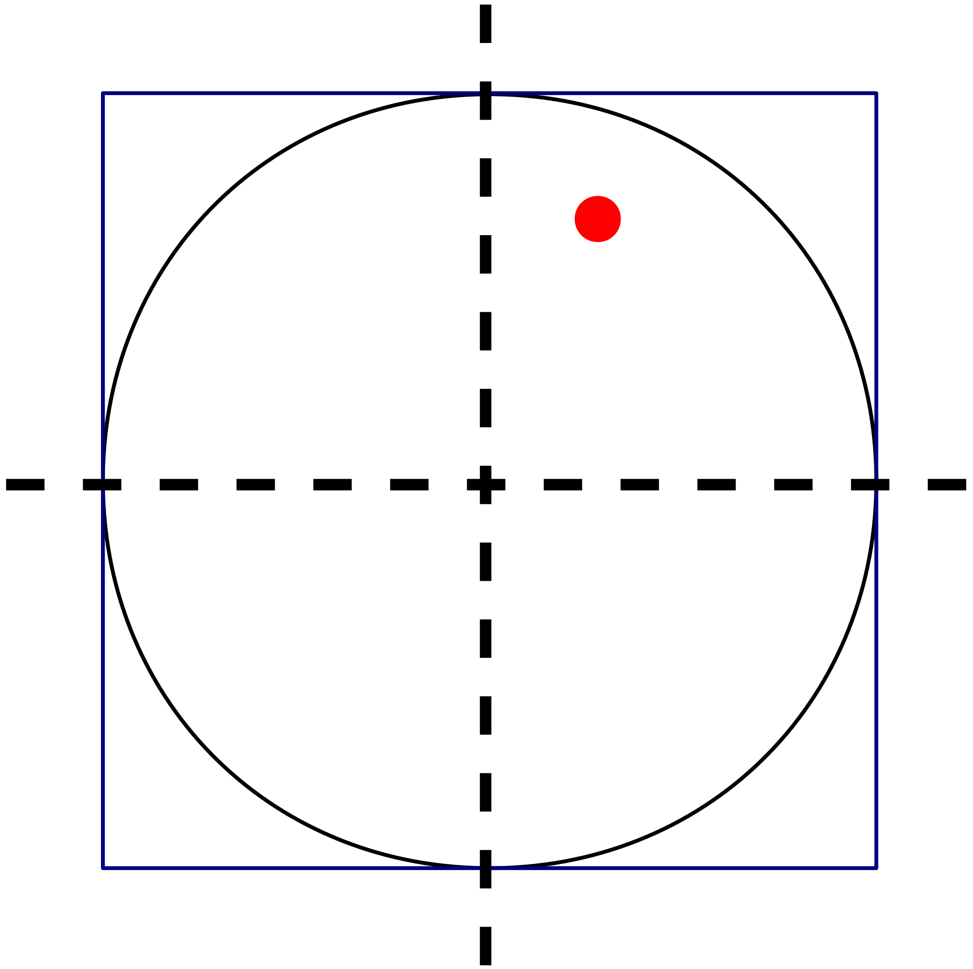
Let us imagine that we have a way of randomly throwing a dot into the square (imagine a game of darts being played, with the square as the board...)



Knowns:

$$r = \sqrt{x^2 + y^2}$$

There is a probability that a uniformly, randomly thrown dot will land in the circle, and a probability that it will land out of the circle. What are those probabilities?

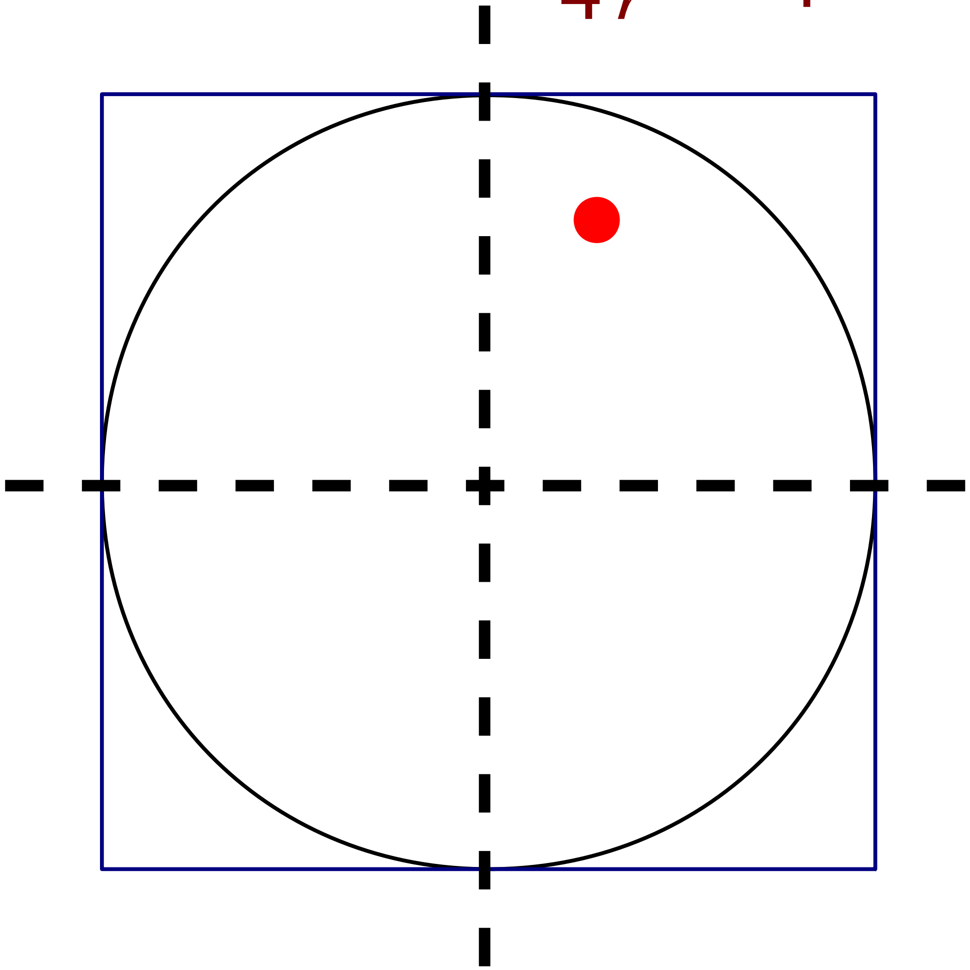


Knowns:

$$r=\sqrt{x^2+y^2}$$

Probability of landing in the circle is merely given by the ratio of the areas of the two objects:

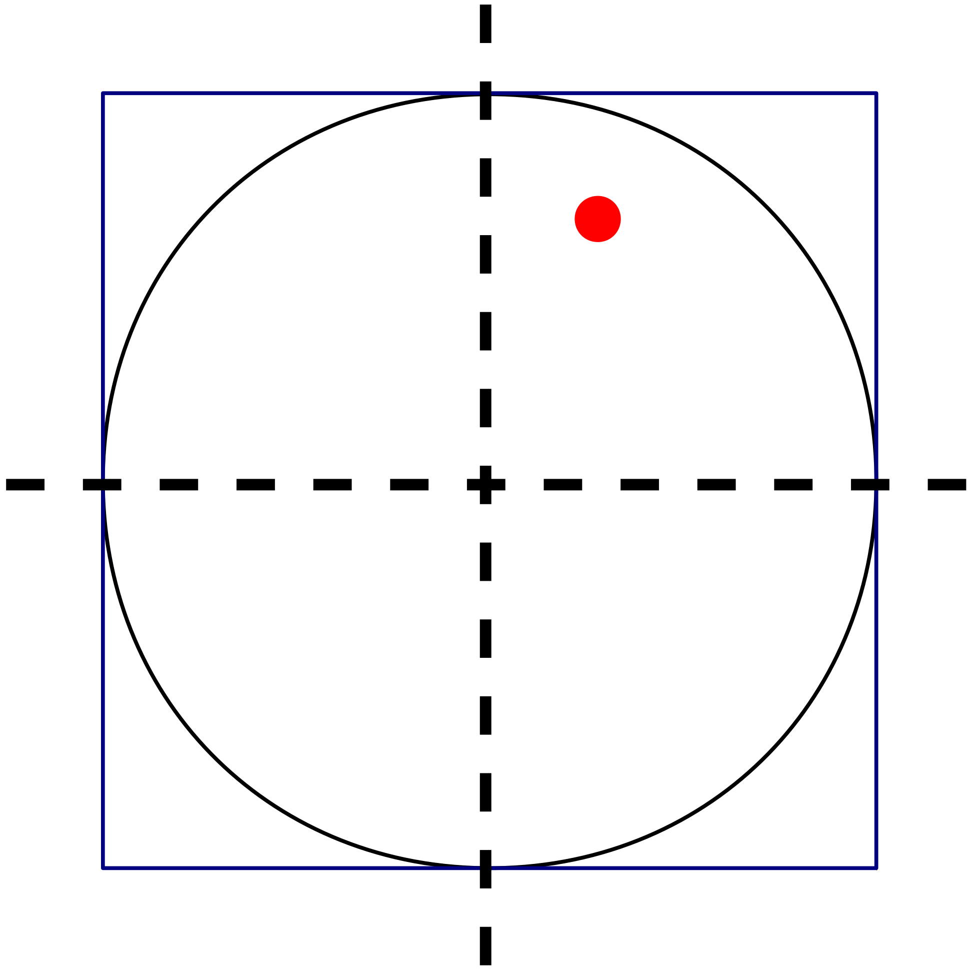
$$P(\mathrm{in}|\mathrm{dot}) = \frac{\pi r^2}{4 r^2} = \frac{\pi}{4}$$



Knowns:

$$r = \sqrt{x^2 + y^2}$$

That's nice – but we're missing a piece . . . just what is that probability on the left side? How can we determine it?

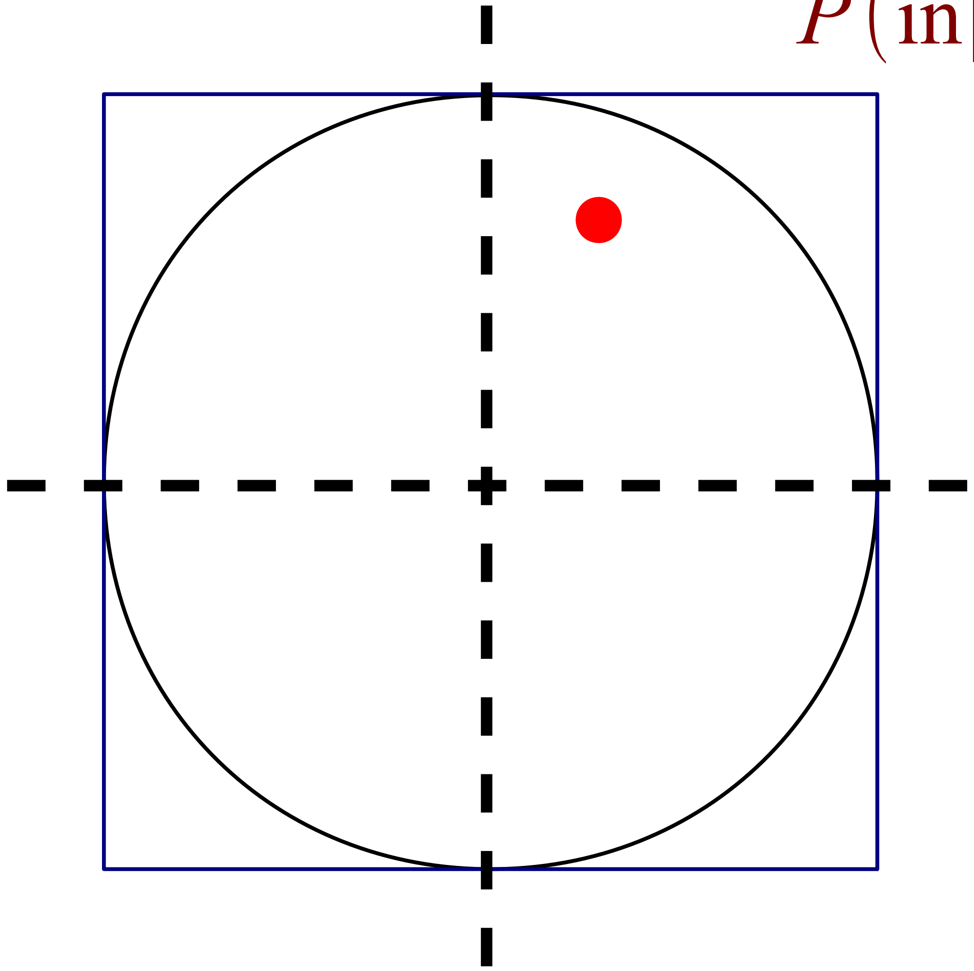


Knowns:

$$r = \sqrt{x^2 + y^2}$$
$$P(\text{in}|\text{dot}) = \frac{\pi}{4}$$

ANSWER: "numerically" - by throwing dots uniformly in the square and counting the number that land inside the circle, divided by the number that we have thrown in total:

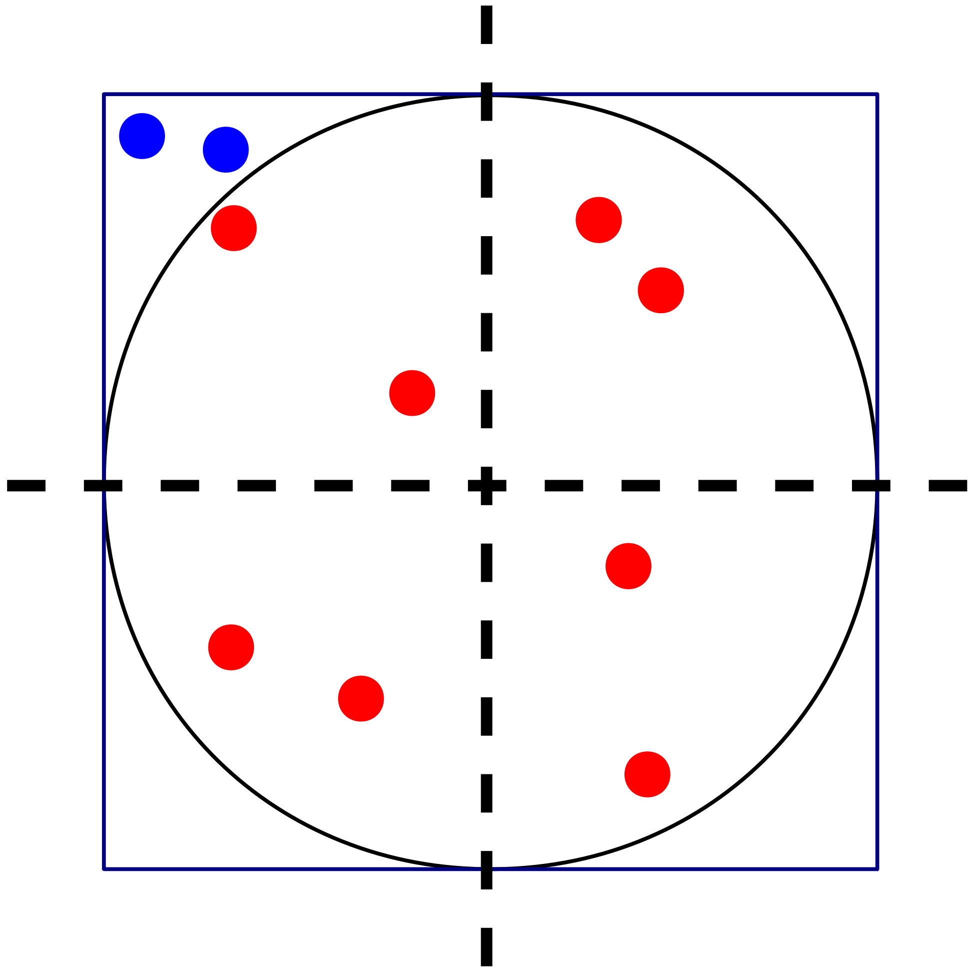
$$P(\mathrm{in}|\mathrm{dot}) = \frac{N_{\mathrm{in}}}{N_{\mathrm{total}}}$$



Knowns:

$$r = \sqrt{x^2 + y^2}$$
$$P(\mathrm{in}|\mathrm{dot}) = \frac{\pi}{4}$$

ANSWER: “numerically” - by throwing dots uniformly in the square and counting the number that land inside the circle, divided by the number that we have thrown in total:



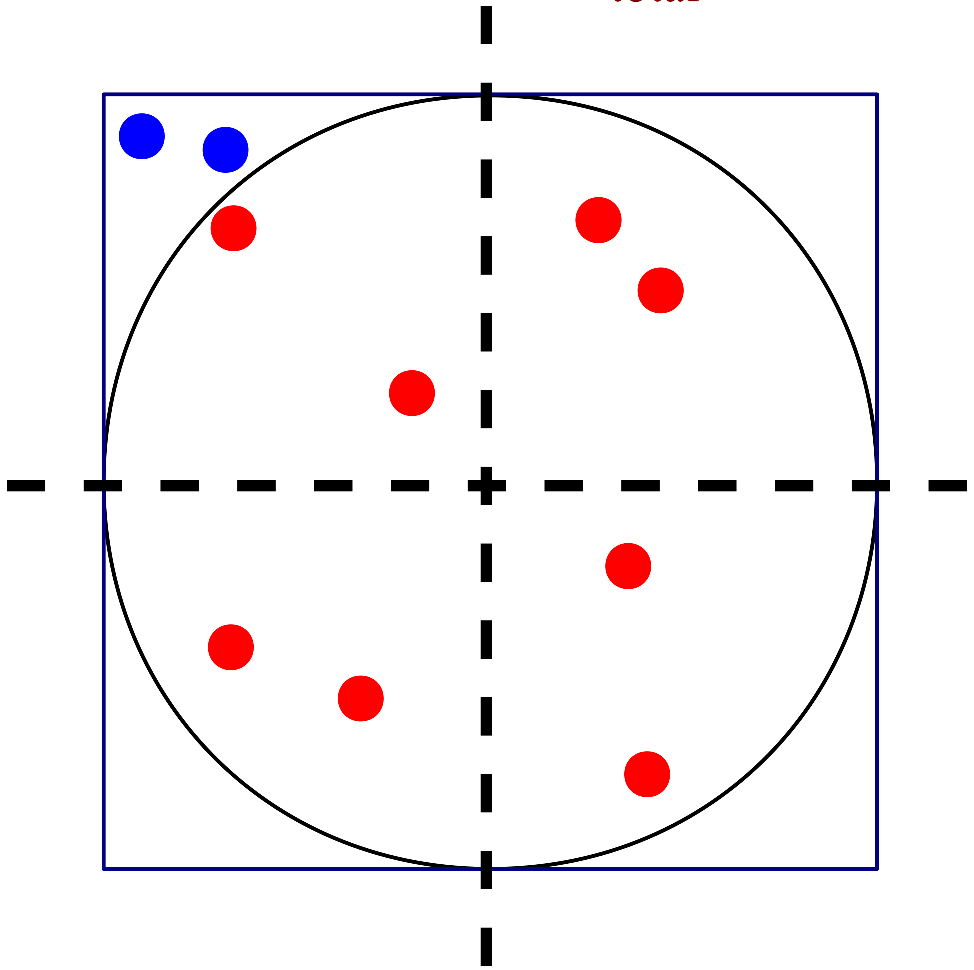
Knowns:

$$r = \sqrt{x^2 + y^2}$$

$$\frac{N_{\text{in}}}{N_{\text{total}}} = \frac{\pi}{4}$$

π is then simply determined numerically via:

$$\pi = 4\frac{N_{\text{in}}}{N_{\text{total}}}$$



Knowns:

$$r = \sqrt{x^2 + y^2}$$

$$\frac{N_{\text{in}}}{N_{\text{total}}} = \frac{\pi}{4}$$

The Pieces

- Random numbers
  - needed to “throw dots” at the board
- Uniformity of coverage
  - we want to pepper the board using uniform random numbers, to avoid creating artificial pileups that create new underlying probabilities
- Code/Programming
  - You can do this manually with a square, an inscribed circle, coordinate axes, and a many-sided die.
  - But that limits your time and precision – computers are faster for such repetitive tasks

Computational Examples

- I will demonstrate the underlying computation framework principles using PYTHON, a free and open-source programming language and computation framework
  - Why? Because every PC in the lab has PYTHON installed and ready to go!
- At the end of this, you will have a program you can take with you and adapt into ANY language.
- If you've never seriously written code before, today is your "lucky" day

Basics of Coding

- Numbers – all programming languages can minimally handle numbers: integers, decimals
- Variables – placeholders for numbers, whose values can be set at any time by the programmer
- Functions – any time you have to repeatedly perform an action, write a function. A “function” is just like in math – it represents a complicated set of actions on variables
- Code – an assembly of variables and functions whose goal is determined by the programmer. “Task-oriented mathematics”
- Coding is the poetry of mathematics – it takes the basic rules of mathematics and does something awesome with them.

Python and Jupyter

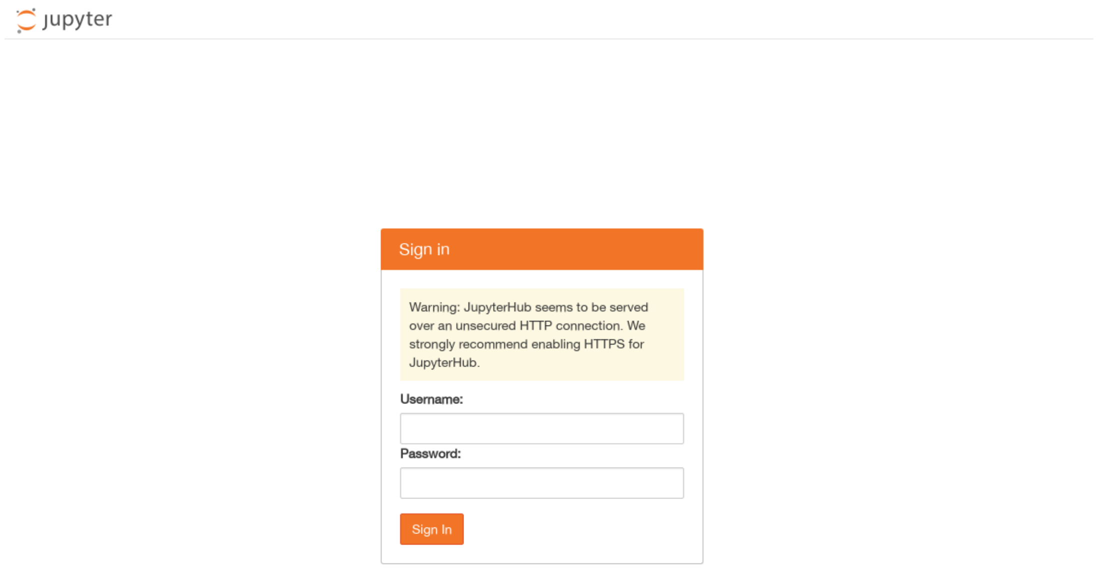
- Python is a programming language
- Jupyter is a framework for developing web-based interactive python software
- We will use Python and Jupyter together today. I think it makes programming more fun and also more share-able.
  - Jupyter notebooks can be shared with other people, who can improve them and share them again.

Using JupyterHub

Open a web browser on your computer. Make sure you are connected to the SMU network (lab PCs already are).

Visit: http://guatemala.physics.smu.edu:8000

You should see something like this:

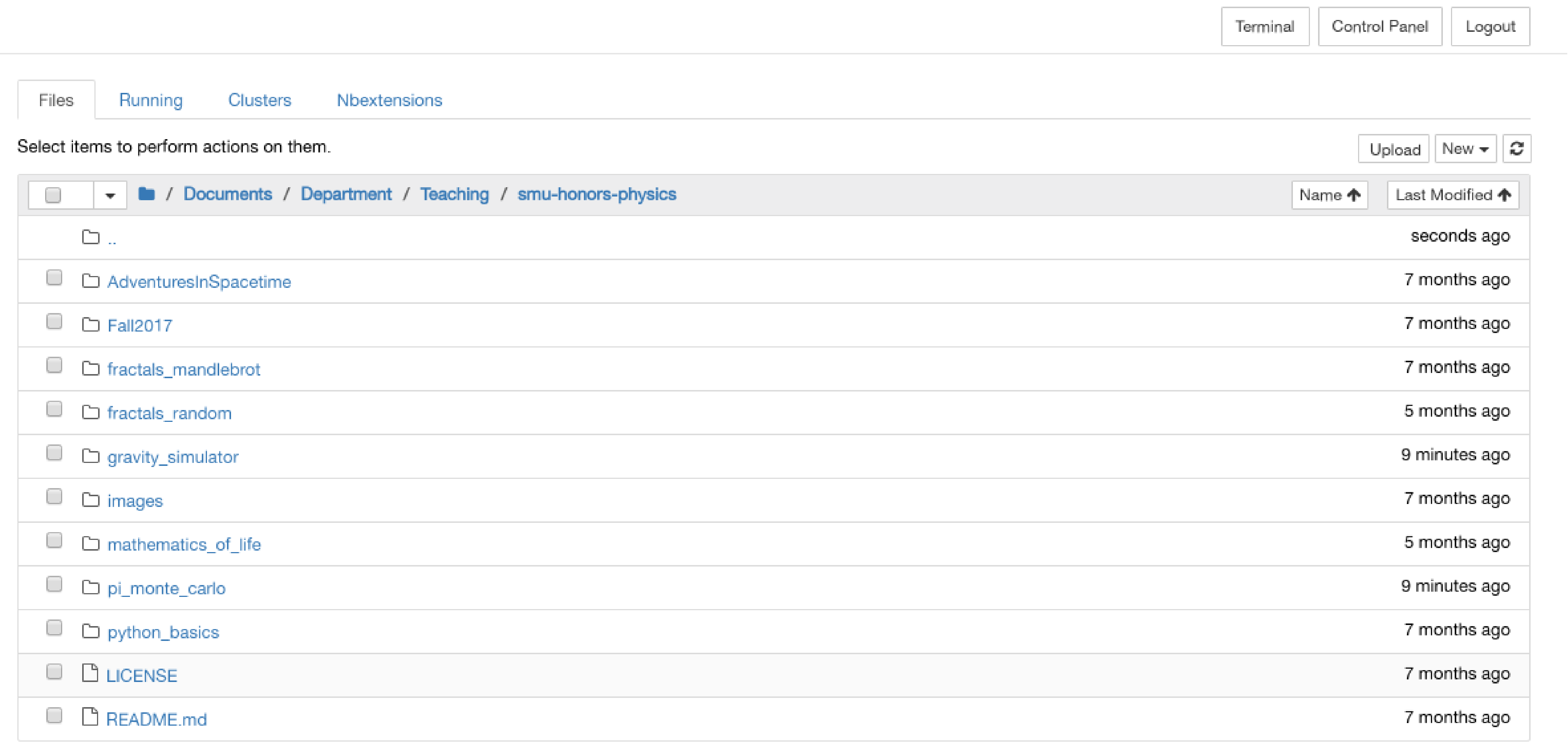


Warning: JupyterHub seems to be served over an unsecured HTTP connection. We strongly recommend enabling HTTPS for JupyterHub.

Username:

Password:

Sign in using the information from the instructor.



Select items to perform actions on them.

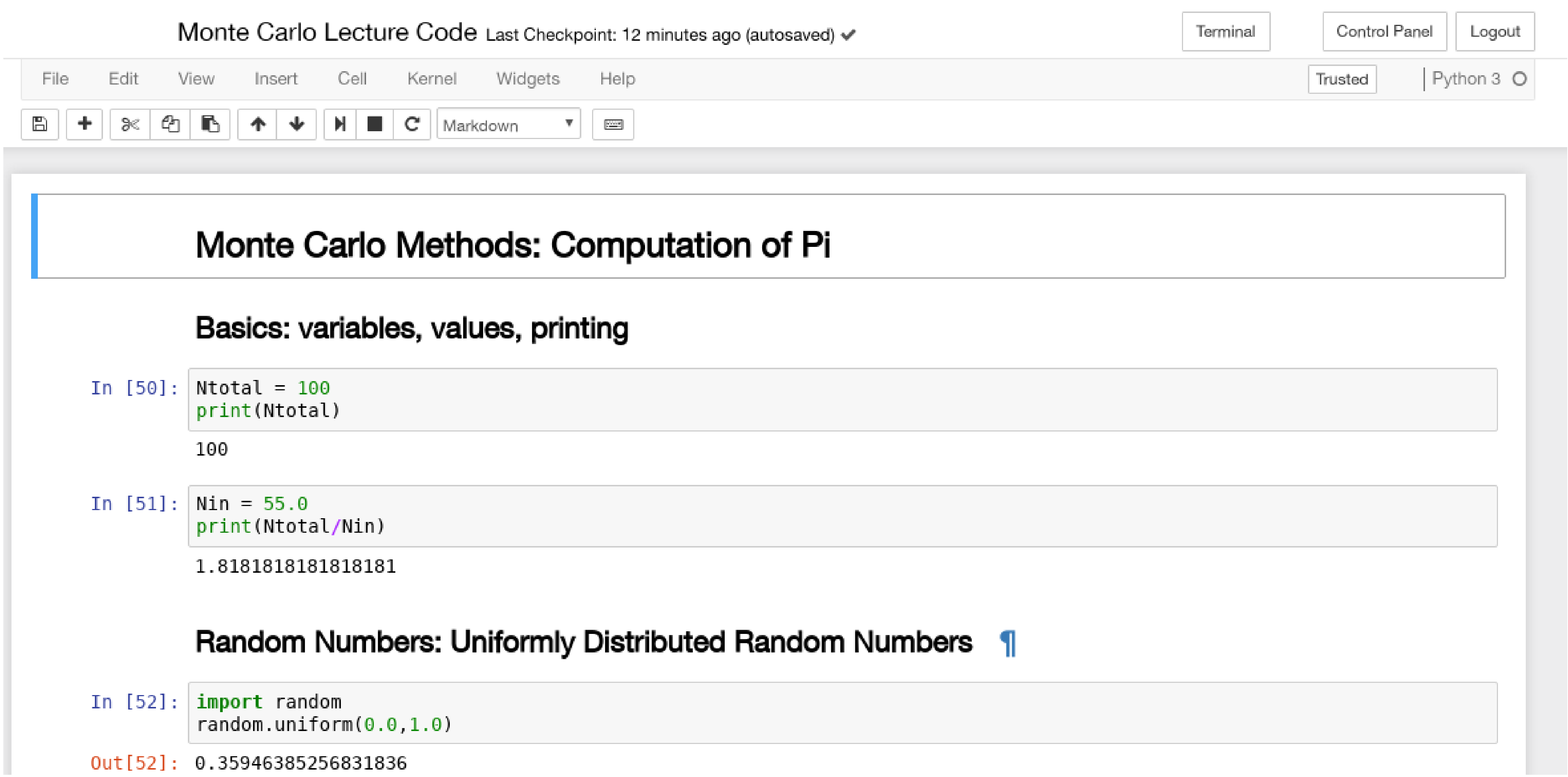
/ Documents / Department / Teaching / smu-honors-physics

..	seconds ago
AdventuresInSpacetime	7 months ago
Fall2017	7 months ago
fractals_mandlebrot	7 months ago
fractals_random	5 months ago
gravity_simulator	9 minutes ago
images	7 months ago
mathematics_of_life	5 months ago
pi_monte_carlo	9 minutes ago
python_basics	7 months ago
LICENSE	7 months ago
README.md	7 months ago

Navigate the folders you see to smu-honors-physics → pi_monte_carlo. Click on the “Monte Carlo Lecture Code.ipynb” notebook.

This code is available anytime for use and editing from:

https://github.com/stephensekula/smu-honors-physics



Monte Carlo Methods: Computation of Pi

Basics: variables, values, printing

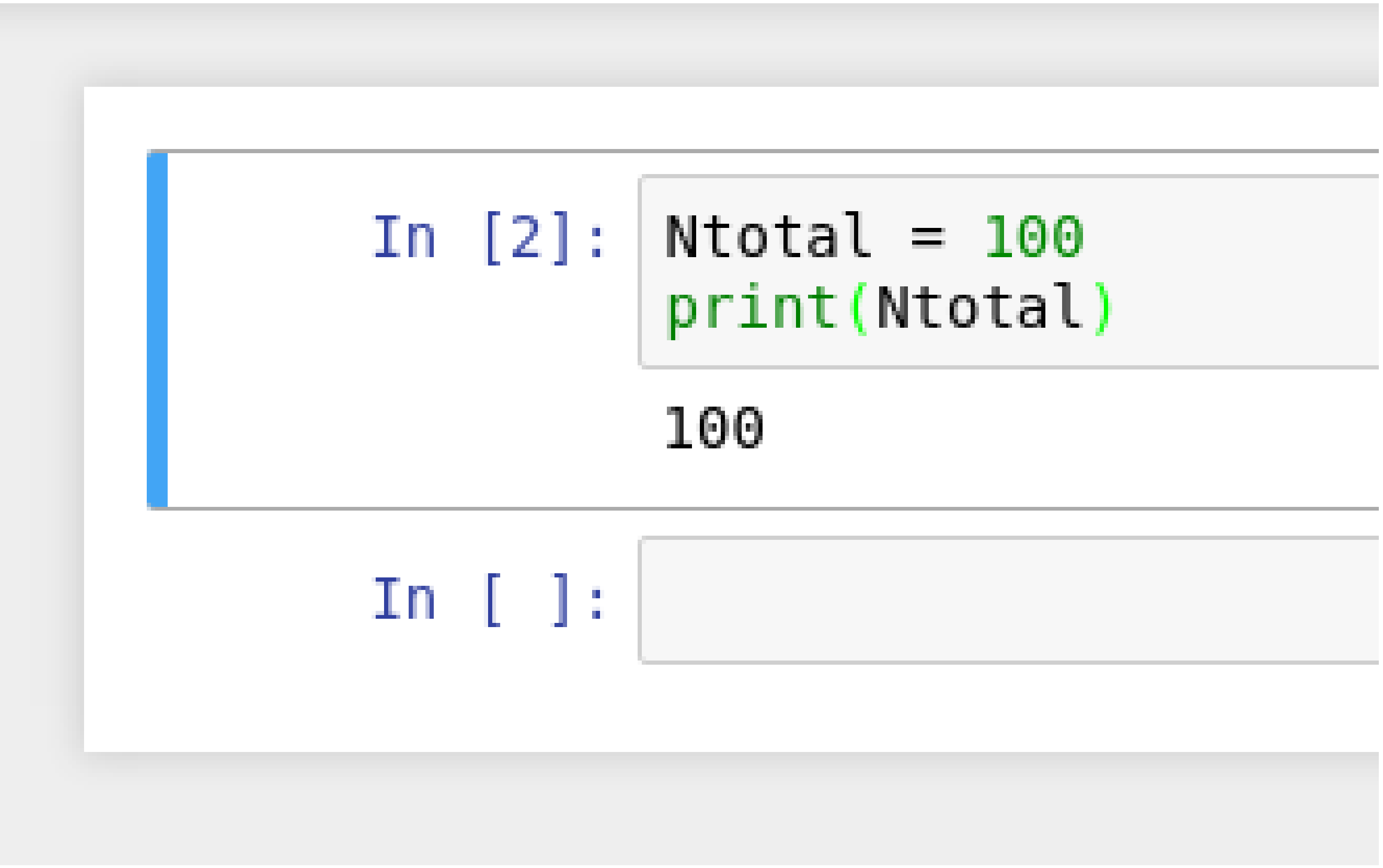
```
In [50]: Ntotal = 100
         print(Ntotal)

         100

In [51]: Nin = 55.0
         print(Ntotal/Nin)

         1.8181818181818181
```

Random Numbers: Uniformly Distributed Random Numbers ¶

```
In [52]: import random
         random.uniform(0.0,1.0)

Out[52]: 0.35946385256831836
```

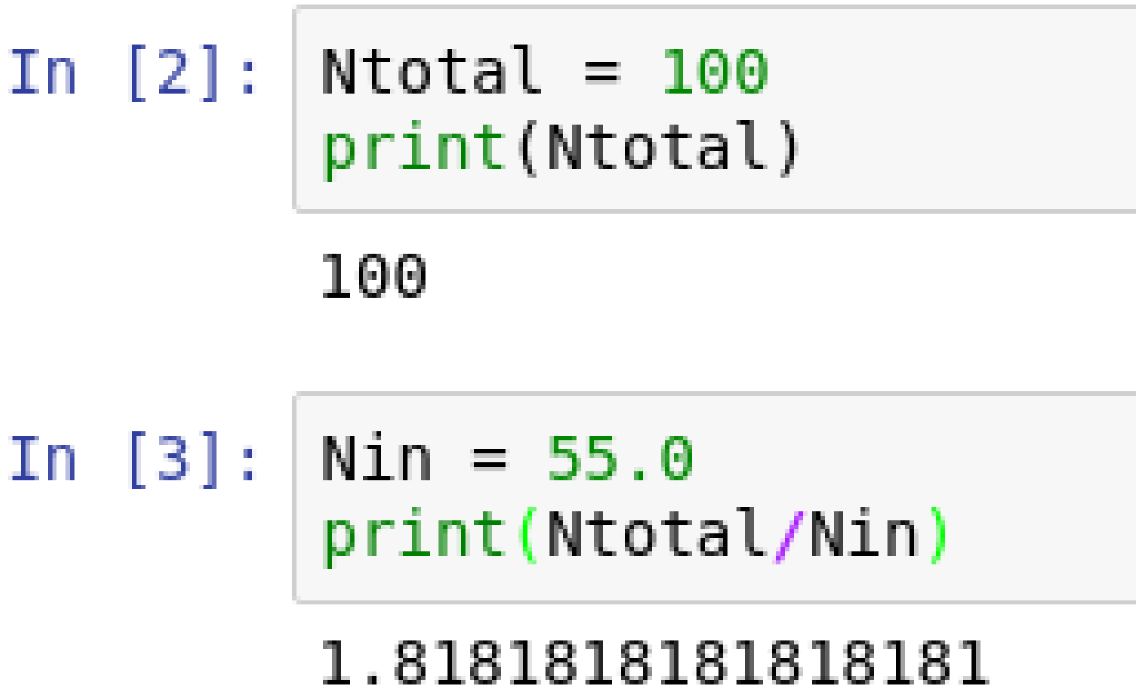
The program and some text and information are available in the notebook. Run the whole notebook by clicking Cell → Run All.

Run an individual cell’s code by clicking on the cell and either pressing the play button on the toolbar at the top or by pressing <SHIFT>+<ENTER>.



```
In [2]: Ntotal = 100
        print(Ntotal)

        100

In [ ]:
```

A simple example of defining a variable ("Ntotal"), setting its value ("=" is the "assignment operator" in Python, and using 100 without a decimal point after the number indicates this is an integer), and then printing its value. To obtain the line-break, press <ENTER>. To execute the whole block of code you just wrote, press <SHIFT+ENTER>.



```
In [2]: Ntotal = 100
        print(Ntotal)

        100

In [3]: Nin = 55.0
        print(Ntotal/Nin)

        1.8181818181818181
```

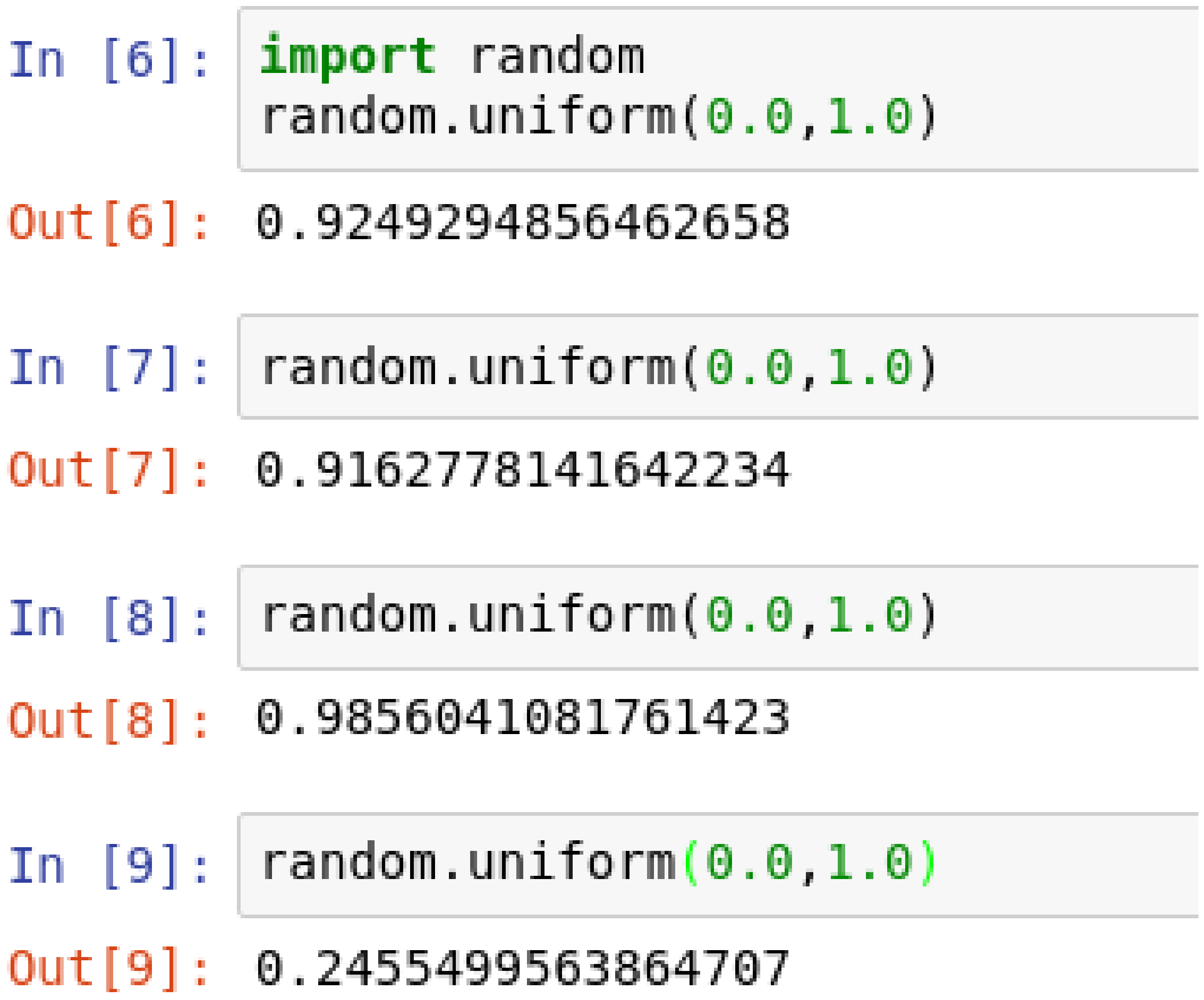
Play with the code. Begin by trying to define or refine a variable in the first block(s) of code (e.g. “Nin”) and set it to 55.0 (decimal points indicate that this number is a “floating point number” - a number with decimal precision). Print the mathematical operation of dividing Ntotal by Nin (“/” is the “divide by” operator in Python).

Note that although Ntotal was an integer and Nin was a floating point number, the math operation return a floating point number.

Uniform Random Numbers

- Computers can generate (pseudo)random numbers using various algorithms
  - this is a whole lecture in and of itself – if you're interested in pseudo-random numbers, etc. go do some independent reading
- We will utilize the PYTHON “random” library and its “.uniform()” function to obtain our uniform random numbers

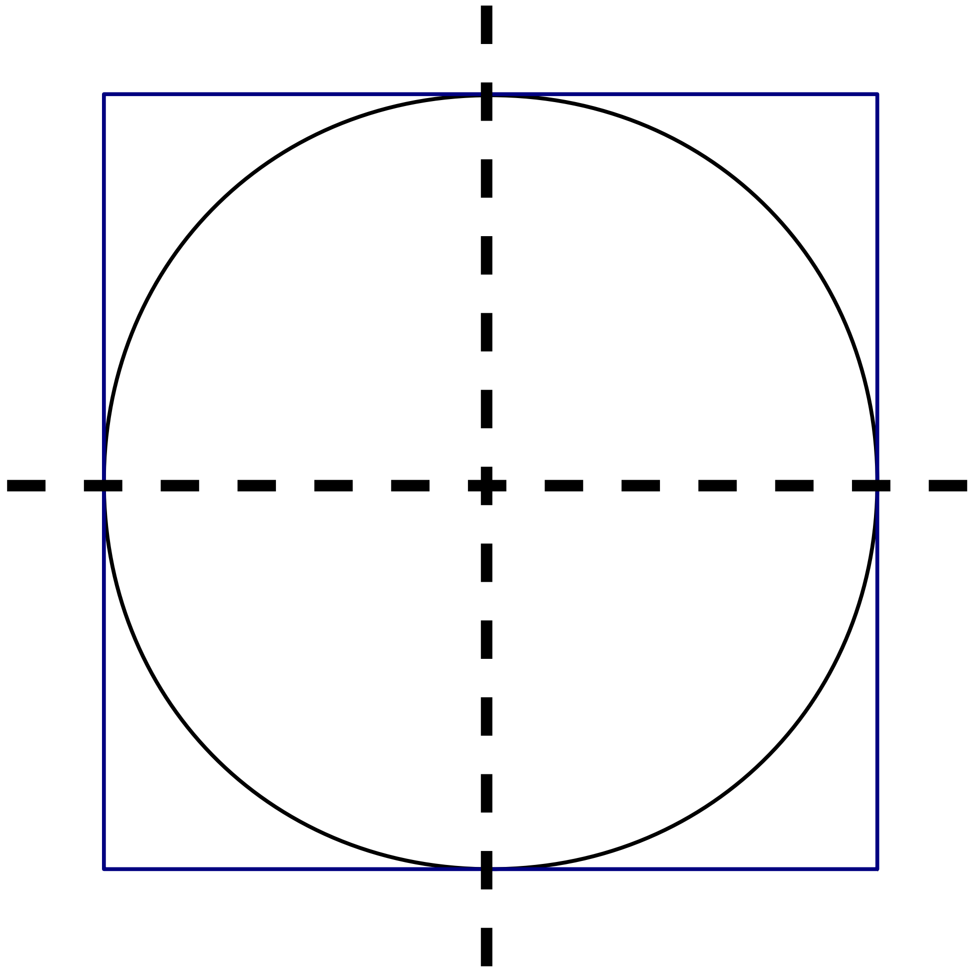
https://docs.python.org/3/library/random.html



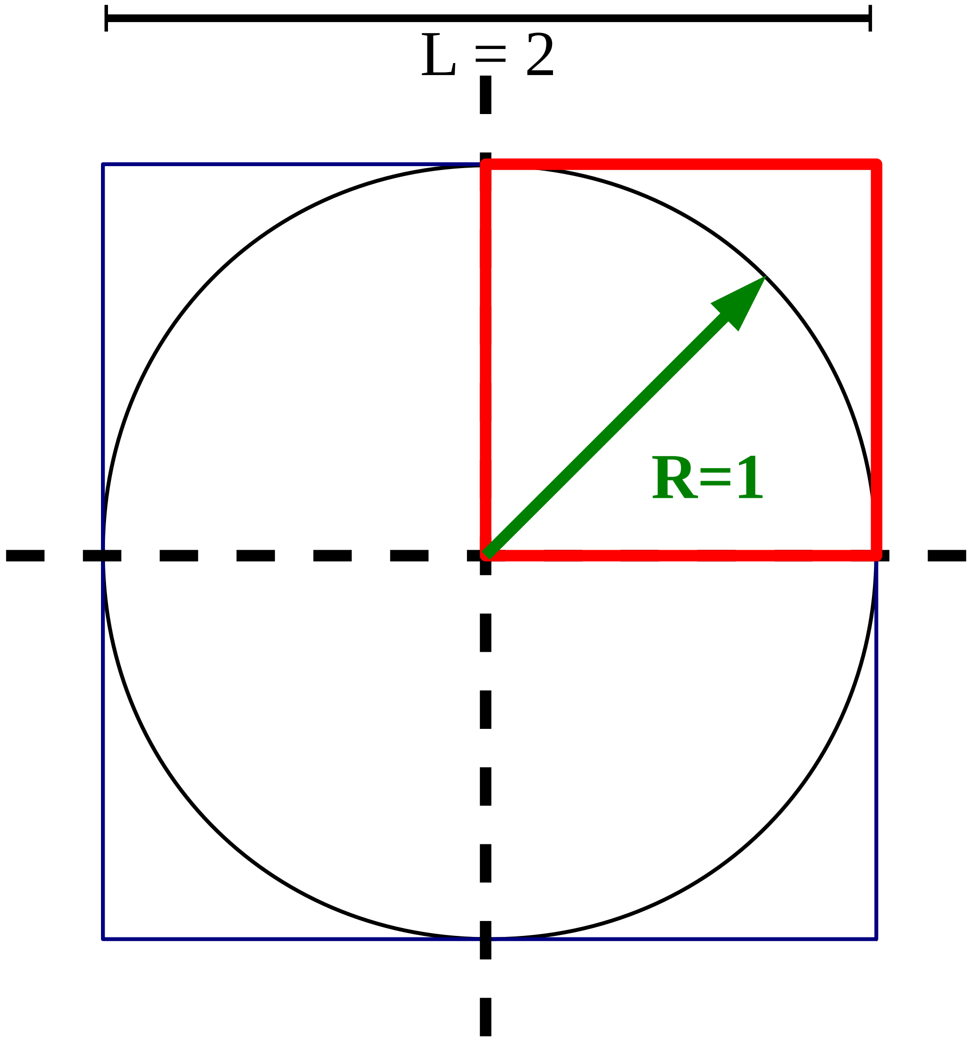
```
In [6]:  import random
         random.uniform(0.0,1.0)

Out[6]:  0.9249294856462658

In [7]:  random.uniform(0.0,1.0)

Out[7]:  0.9162778141642234

In [8]:  random.uniform(0.0,1.0)

Out[8]:  0.9856041081761423

In [9]:  random.uniform(0.0,1.0)

Out[9]:  0.2455499563864707
```

“random.uniform(0.0,1.0)” generates a uniform random floating-point decimal number between 0 and 1 (inclusive)

Designing our "game board"



Designing our “game board”

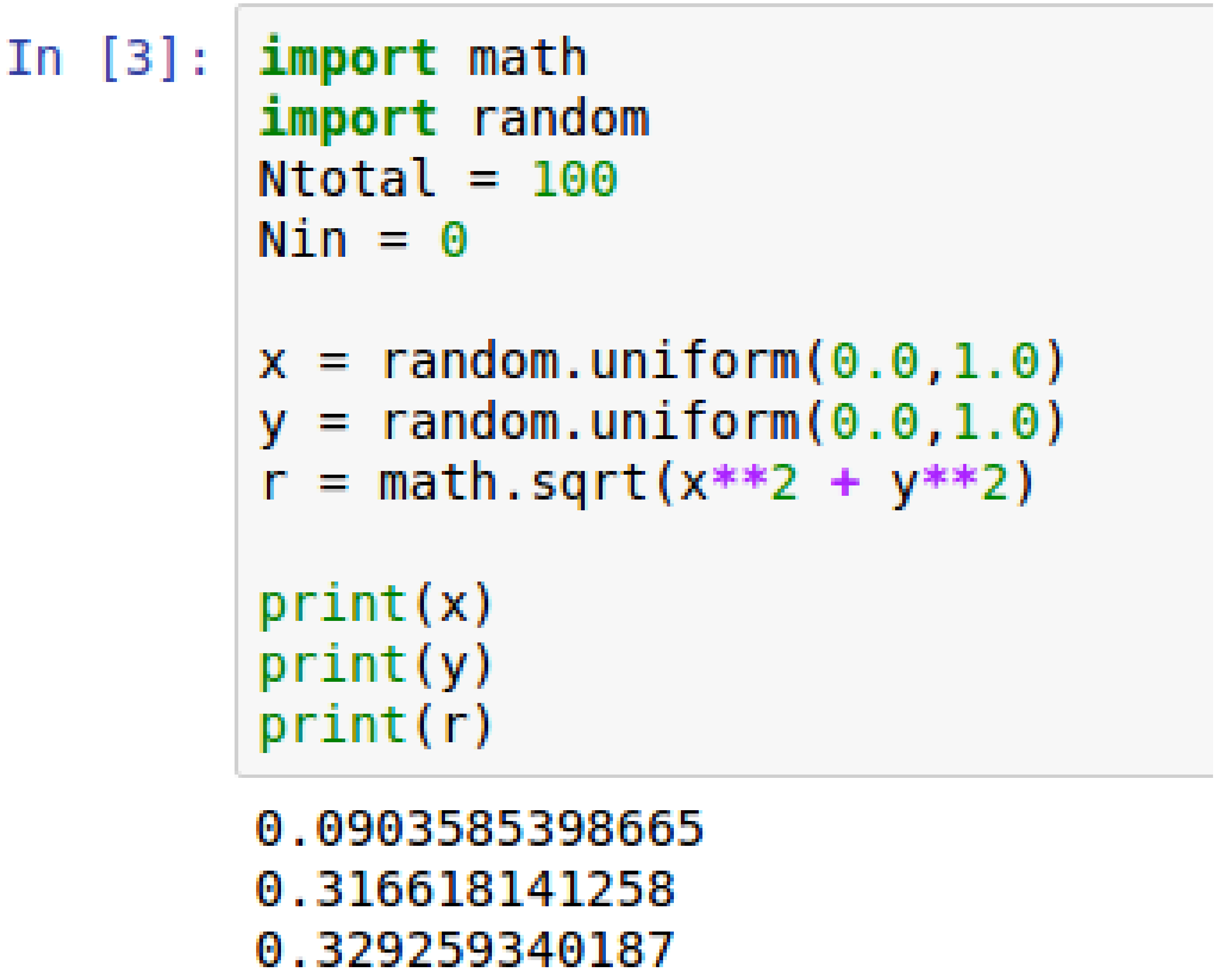


We don't need the whole game board – we can just use one-quarter of it. This keeps the program simple!

Alternative: you can redefine the range of “uniform()” to generate random numbers between -1 and 1

$$\frac{(1/4)\pi r^2}{(1/4)4r^2}=\frac{\pi}{4}=P(\mathrm{in}|\mathrm{dot})$$

https://docs.python.org/3/library/math.html



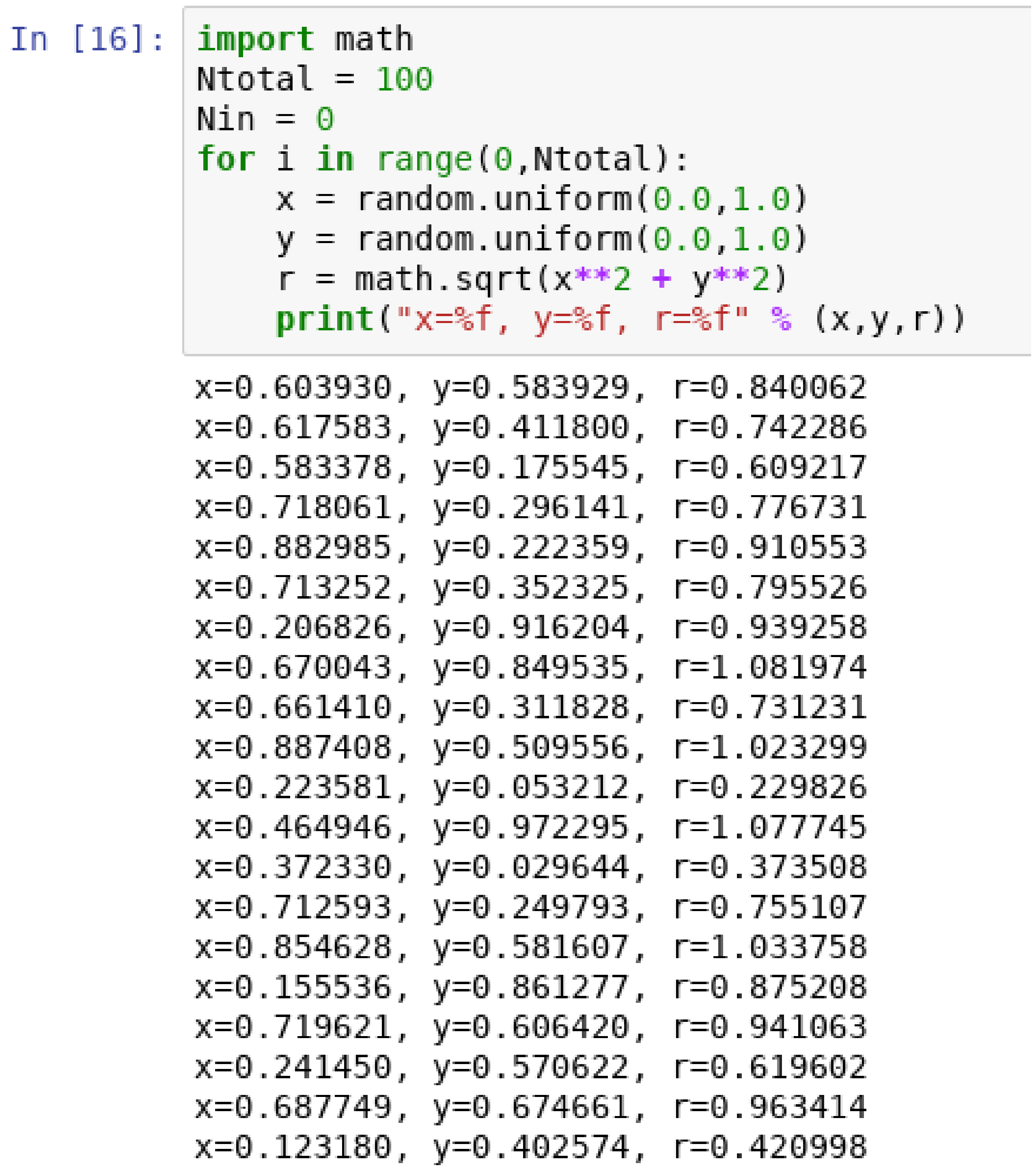
```
In [3]: import math
        import random
        Ntotal = 100
        Nin = 0

        x = random.uniform(0.0,1.0)
        y = random.uniform(0.0,1.0)
        r = math.sqrt(x**2 + y**2)

        print(x)
        print(y)
        print(r)
```

```
0.0903585398665
0.316618141258
0.329259340187
```

Repetition

- You don't want to manually type 100 (or more) computations of your dot throwing
- You need a loop!
- A “loop” is a small structure that automatically repeats your computation a specified number of times
- In PYTHON:

```
Ntotal = 100
Nin = 0
for i in range(0,Ntotal):
    x = random.uniform(0.0,1.0)
    y = random.uniform(0.0,1.0)
```



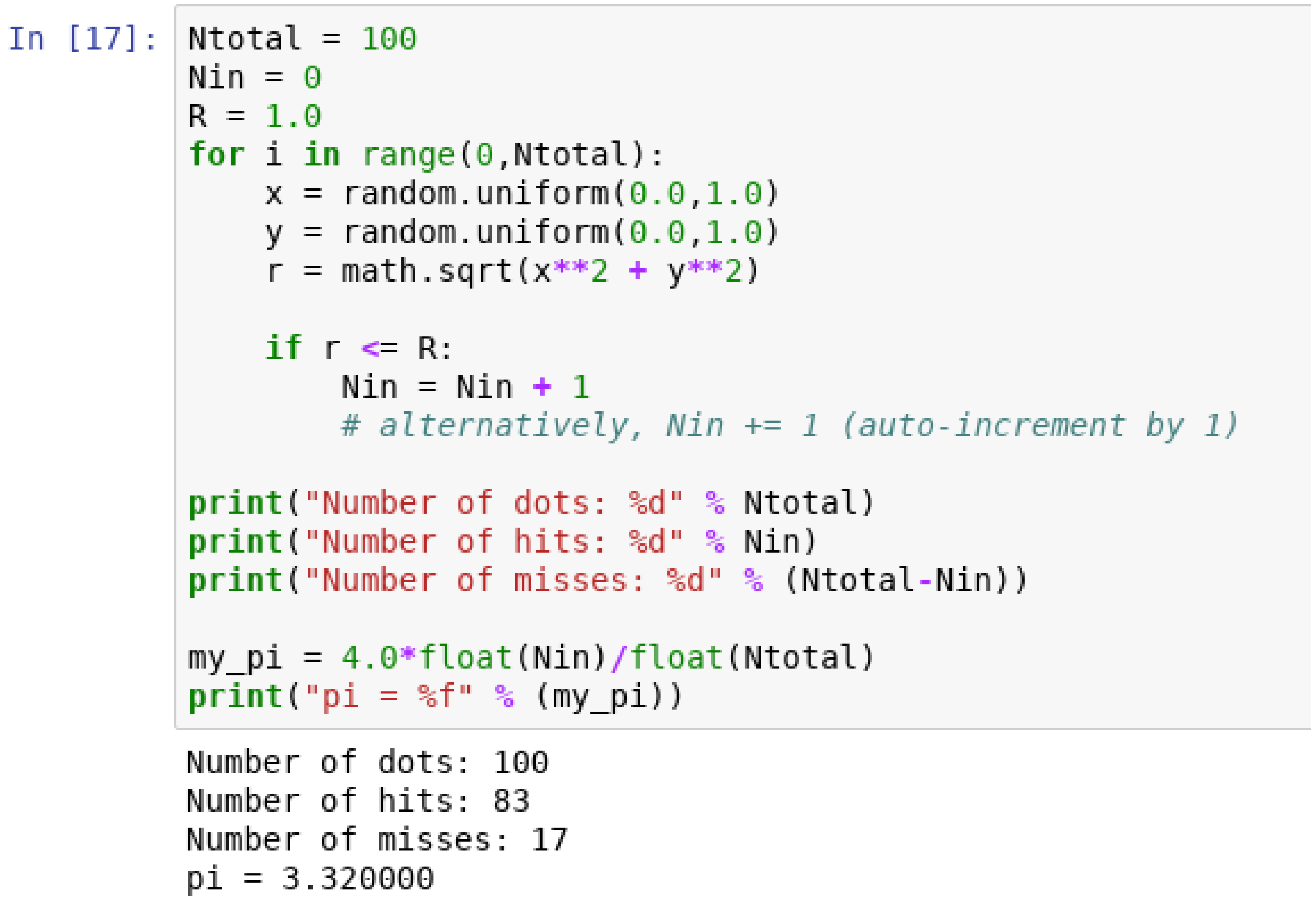
```
In [16]: import math
         Ntotal = 100
         Nin = 0
         for i in range(0,Ntotal):
             x = random.uniform(0.0,1.0)
             y = random.uniform(0.0,1.0)
             r = math.sqrt(x**2 + y**2)
             print("x=%f, y=%f, r=%f" % (x,y,r))
```

```
x=0.603930, y=0.583929, r=0.840062
x=0.617583, y=0.411800, r=0.742286
x=0.583378, y=0.175545, r=0.609217
x=0.718061, y=0.296141, r=0.776731
x=0.882985, y=0.222359, r=0.910553
x=0.713252, y=0.352325, r=0.795526
x=0.206826, y=0.916204, r=0.939258
x=0.670043, y=0.849535, r=1.081974
x=0.661410, y=0.311828, r=0.731231
x=0.887408, y=0.509556, r=1.023299
x=0.223581, y=0.053212, r=0.229826
x=0.464946, y=0.972295, r=1.077745
x=0.372330, y=0.029644, r=0.373508
x=0.712593, y=0.249793, r=0.755107
x=0.854628, y=0.581607, r=1.033758
x=0.155536, y=0.861277, r=0.875208
x=0.719621, y=0.606420, r=0.941063
x=0.241450, y=0.570622, r=0.619602
x=0.687749, y=0.674661, r=0.963414
x=0.123180, y=0.402574, r=0.420998
```

“Loops” are powerful – they are a major workhorse of any repetitive task coded up in a programming language.

Final Piece

- So we have generated a dot by generating its x and y coordinates throwing uniform random numbers...
- How do we determine if it's “in” or “out” of the circle?
- ANSWER:
  - if $r = \sqrt{(x^2+y^2)} \leq R$, it's in or on the circle; otherwise, it is out of the circle!



```
In [17]: Ntotal = 100
         Nin = 0
         R = 1.0
         for i in range(0,Ntotal):
             x = random.uniform(0.0,1.0)
             y = random.uniform(0.0,1.0)
             r = math.sqrt(x**2 + y**2)

             if r <= R:
                 Nin = Nin + 1
                 # alternatively, Nin += 1 (auto-increment by 1)

         print("Number of dots: %d" % Ntotal)
         print("Number of hits: %d" % Nin)
         print("Number of misses: %d" % (Ntotal-Nin))

         my_pi = 4.0*float(Nin)/float(Ntotal)
         print("pi = %f" % (my_pi))
```

```
Number of dots: 100
Number of hits: 83
Number of misses: 17
pi = 3.320000
```

A working program.

You can increase N_{total} to get increased precision!

A Comment on Precision

- Given finite statistics, each set of trials carries an uncertainty ($\pi \pm \sigma_\pi$). A point, (x,y), can either be in/on or out of the circle of radius, R. Thus the uncertainty on N_{in} can be treated as a binomial error:

$$\sigma_{N_{\text{in}}} = \sqrt{N_{\text{total}} \cdot p(1-p)} \text{ where } p = N_{\text{in}} / N_{\text{total}}$$

- Propagating this to π:

$$\sigma_\pi = 4 \cdot \sigma_{N_{\text{in}}} / N_{\text{total}} = 4 \sqrt{\frac{N_{\text{in}}}{N_{\text{total}}^2}\left(1 - \frac{N_{\text{in}}}{N_{\text{total}}}\right)}$$

Precision (continued)

- Relative error:

$$\frac{\sigma_\pi}{\pi} = \sqrt{\frac{1}{N_{\text{in}}} - \frac{1}{N_{\text{total}}}}$$

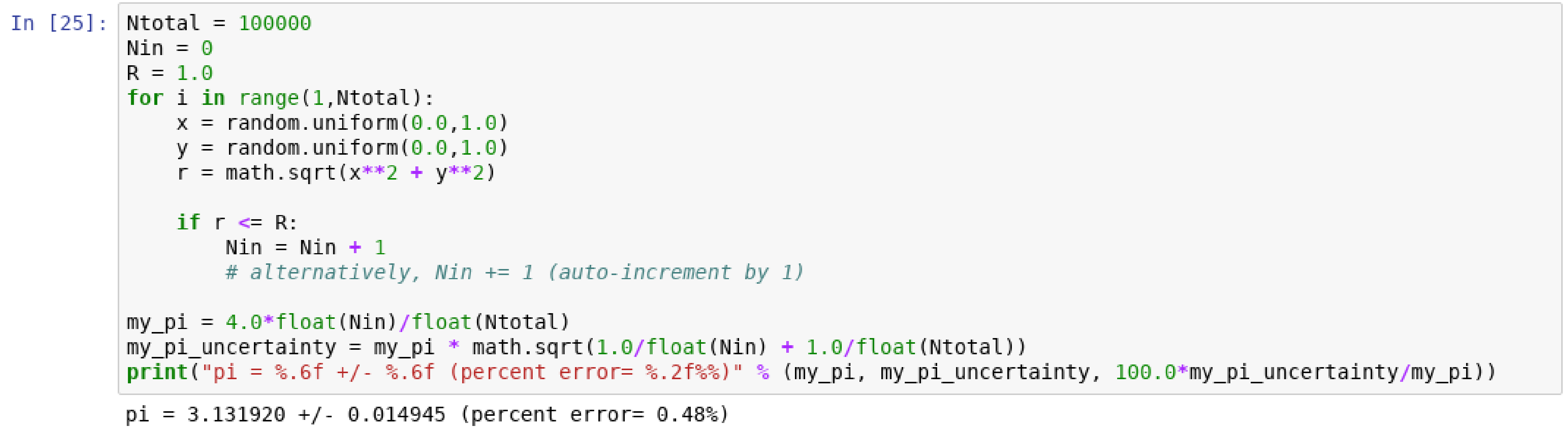
- For 100 trials, σ_π/π = 15.2% (e.g. 3.04 ± 0.46)
- For 1000 trials, σ_π/π = 4.8% (e.g. 3.15 ± 0.15)
- For 10,000 trials: σ_π/π = 1.5% (e.g. 3.141 ± 0.047)
- For 100,000 trials: σ_π/π = 0.48% (e.g. 3.128 ± 0.015)

improves by √10=3.1623

improves by √10=3.1623

improves by √10=3.1623

Note that uncertainty scales only as $1/\sqrt{N_{\text{total}}}$

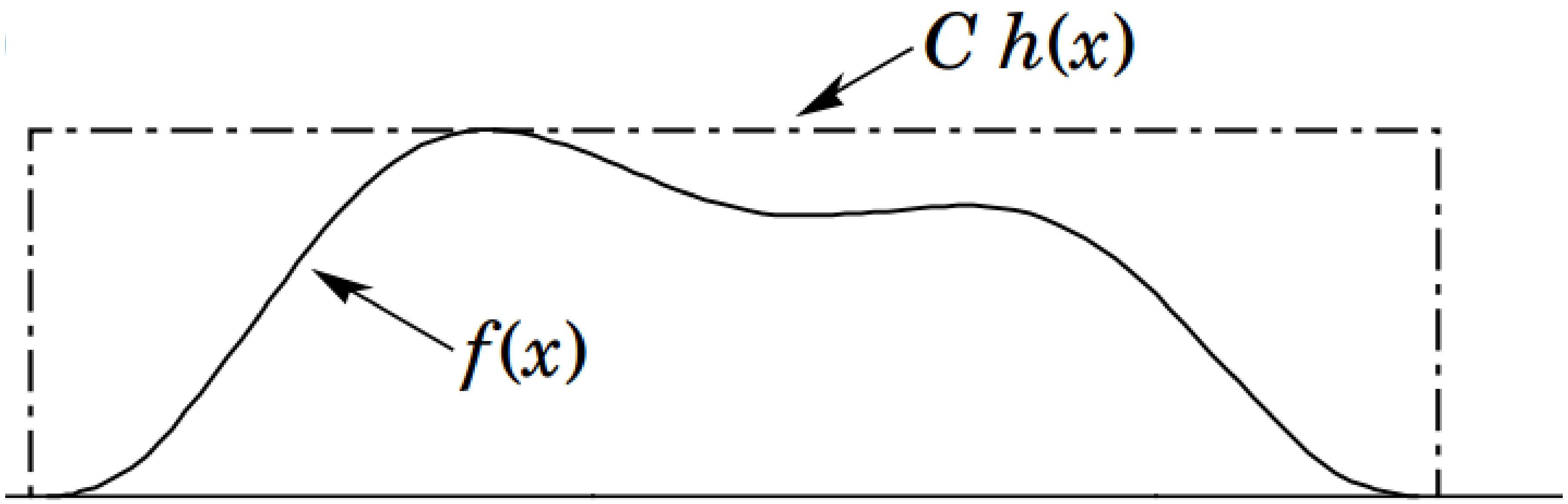


```
In [25]: Ntotal = 100000
         Nin = 0
         R = 1.0
         for i in range(1,Ntotal):
             x = random.uniform(0.0,1.0)
             y = random.uniform(0.0,1.0)
             r = math.sqrt(x**2 + y**2)

             if r <= R:
                 Nin = Nin + 1
                 # alternatively, Nin += 1 (auto-increment by 1)

         my_pi = 4.0*float(Nin)/float(Ntotal)
         my_pi_uncertainty = my_pi * math.sqrt(1.0/float(Nin) + 1.0/float(Ntotal))
         print("pi = %.6f +/- %.6f (percent error= %.2f%%)" % (my_pi, my_pi_uncertainty, 100.0*my_pi_uncertainty/my_pi))

         pi = 3.131920 +/- 0.014945 (percent error= 0.48%)
```

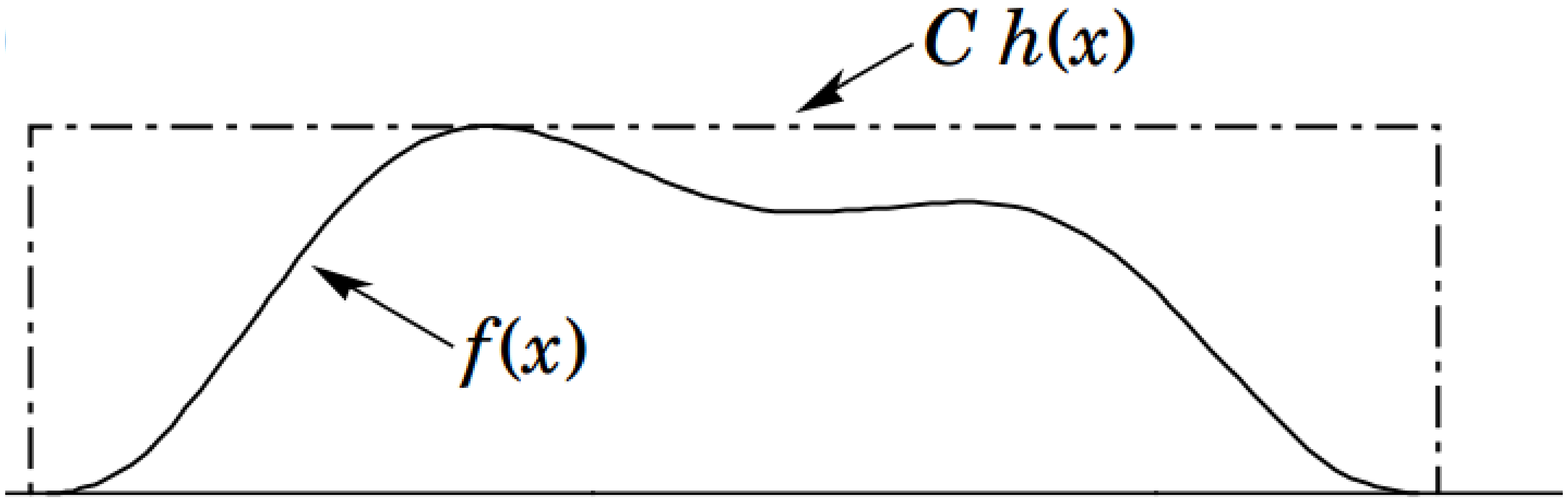
Play around with computing all kinds of things, like the percent error, the absolute error, and printing extended and nicely formatted numbers. Python is fun! Imagine if you had learned this earlier – how much better a homework result could you have prepared by attacking problems both analytically AND numerically?

Why is this powerful?

- You've just learned how to compute an integral NUMERICALLY.
- You can apply this technique to any function whose integral (area) you wish to determine
- For instance, consider the next slide.



- Given an arbitrary function, f(x), you can determine its integral numerically using the "Accept/Reject Method"
- **First**, find the maximum value of the function (e.g. either analytically, if you like, or by calculating the value of f(x) over steps in x to find the max. value, which I denote F(x))
- **Second**, enclose the function in a box, h(x), whose height is F(x) and whose length encloses as much of f(x) as is possible.
- **Third**, compute the area of the box (easy!)
- **Fourth**, throw points in the box using uniform random numbers. Throw a value for x, denotes x'. Throw a value for y, denoted y'. If y' < f(x'), it's a hit! If not, it's a miss!

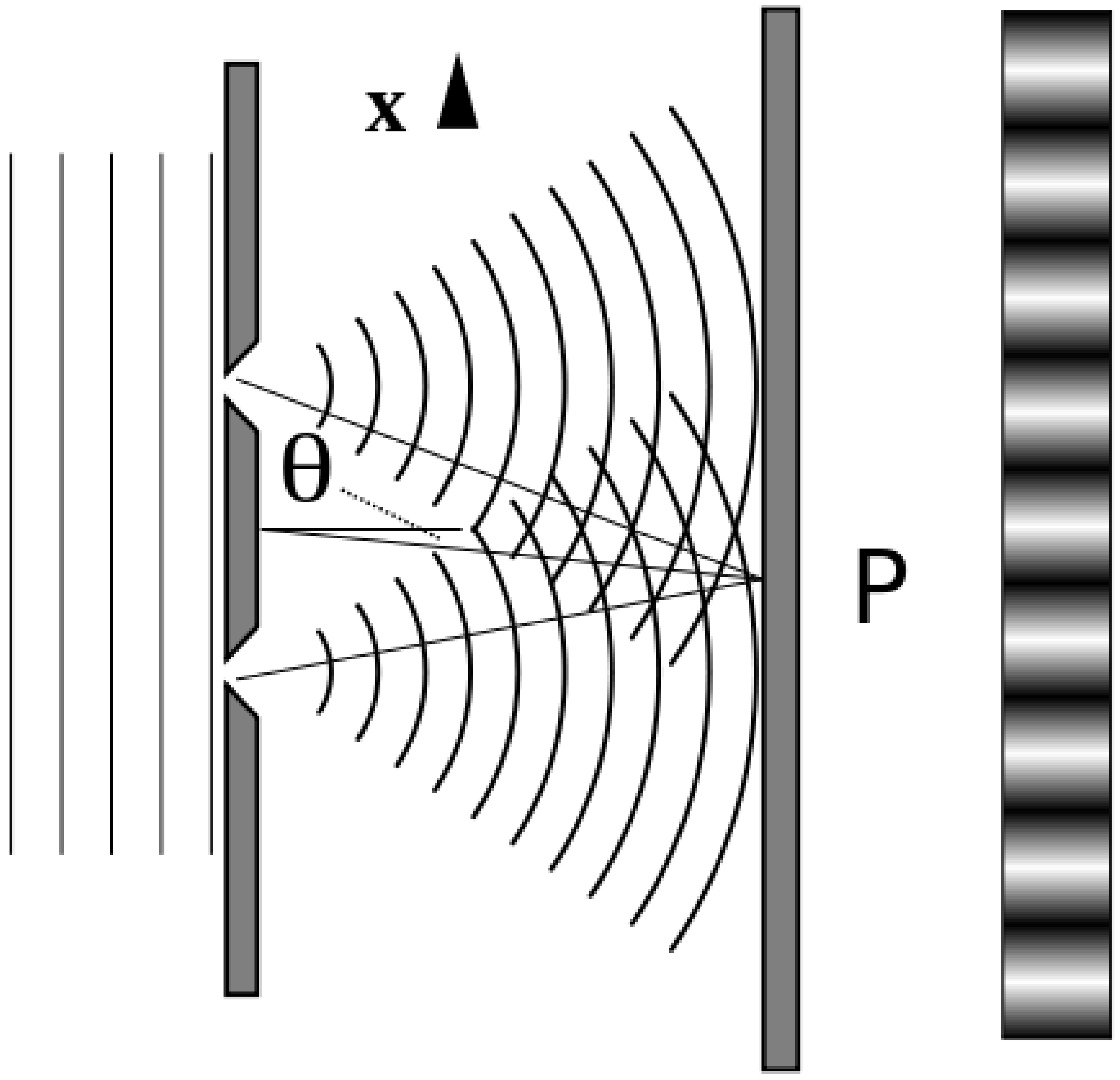


$$\frac{N_{\text{hits}}}{N_{\text{total}}}=\frac{I\left(f\left(x\right)\right)}{A\left(h\left(x\right)\right)}$$

This, in the real world, is how physicists, engineers, statisticians, mathematicians, etc. compute integrals of arbitrary functions.
Learn it. Love it. It will save you.

Generating Simulations

- The Monte Carlo technique, given a function that represents the probability of an outcome, can be used to generate “simulated data”
- Simulated data is useful in designing an experiment, or even “running” an experiment over and over to see all possible outcomes


x
θ
P

Young's Double-Slit Experiment Simulation

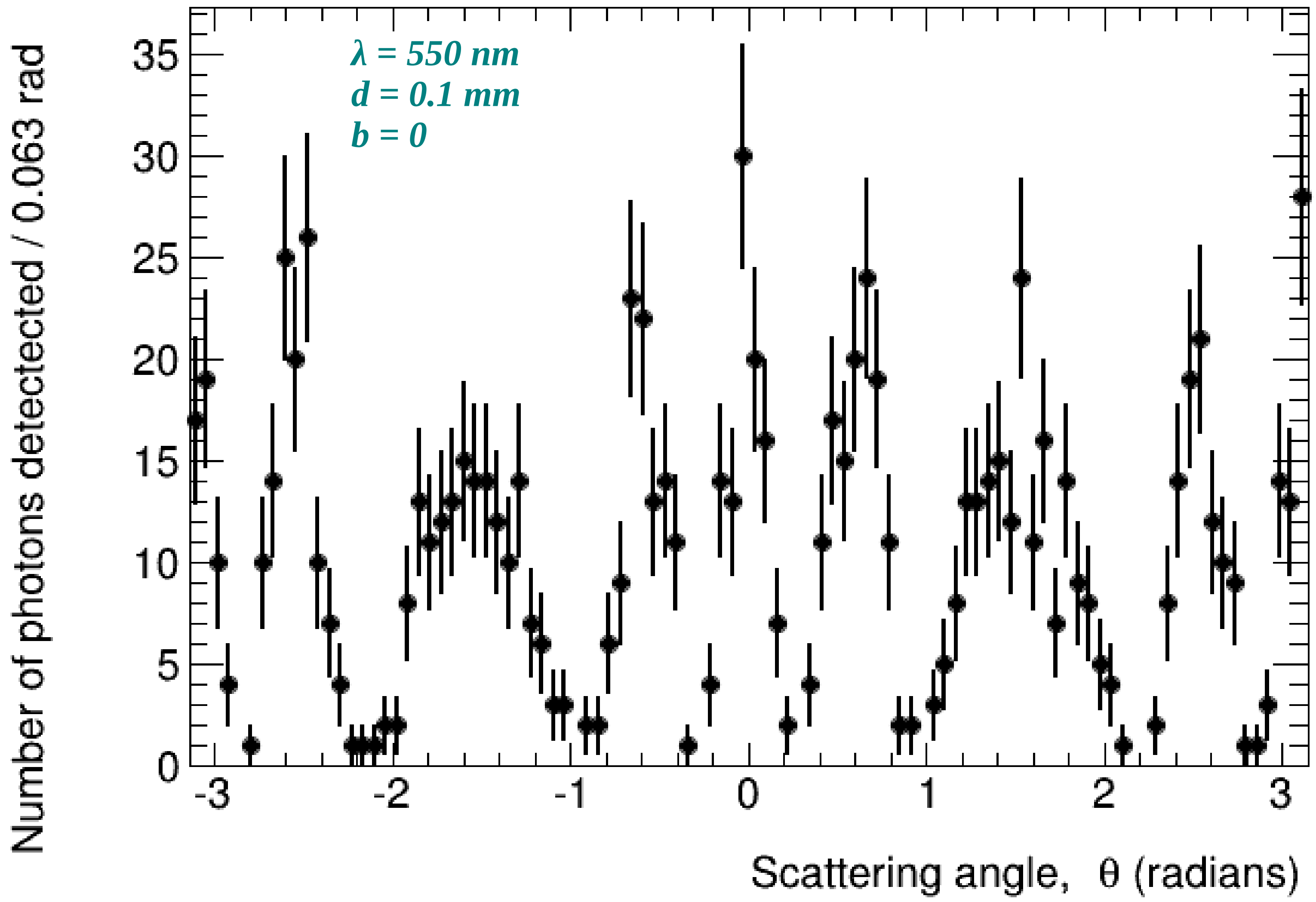
- Consider slits of width, b, separated by a distance, d.
- Given the function that describes the probability of finding a photon at a given angle:

$$I(\theta) \propto \cos^2\left[\frac{\pi d \sin(\theta)}{\lambda}\right] \operatorname{sinc}^2\left[\frac{\pi b \sin(\theta)}{\lambda}\right]$$

$$\operatorname{sinc}(x) = \begin{cases} \sin(x)/x & (x \neq 0) \\ 1 & (x = 0) \end{cases}$$

Next Steps

- Need the max. value of I(θ)
  - occurs at θ = 0
- Use that to compute the height of the box; the width of the box is 2π (ranging from -π to +π)
- “Throw” random points in the box until you get 1000 “accepts”
- Now you have a “simulated data” sample of 1000 photons scattered in the two-slit experiment.



1000 simulated photons scattered through a double-slit experiment. This was done in C++ using the free ROOT High-Energy Physics data analysis framework, so I could easily generate a *histogram* – a binned data sample.

Resources

- SMU Honors Physics Software Project
  https://github.com/stephensekula/smu-honors-physics
- Python: (open-source, free)
  http://www.python.org/
- Project Jupyter: (open-source, free)
  http://jupyter.org/
- Mathematica: (closed-source, non-free)
  http://www.wolfram.com/mathematica/
- Maxima (open-source, free “Mathematica”)
  http://maxima.sourceforge.net
- Monte Carlo Techniques:
  http://en.wikipedia.org/wiki/Monte_Carlo_method