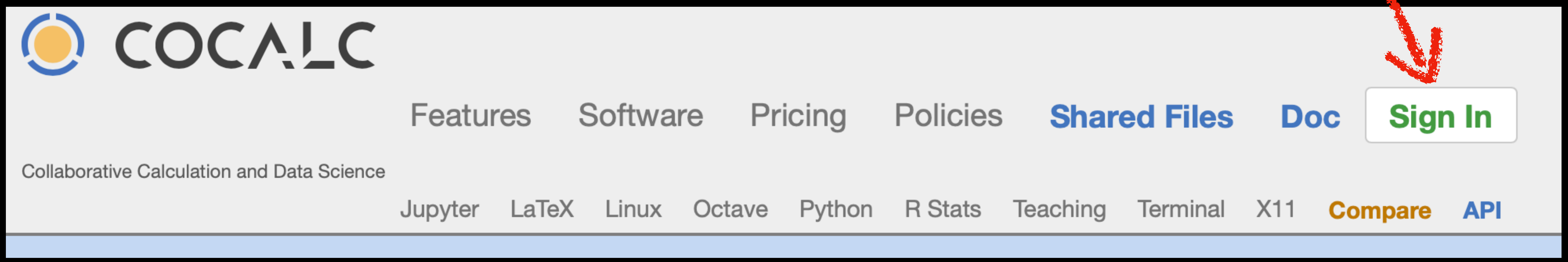
Day 2

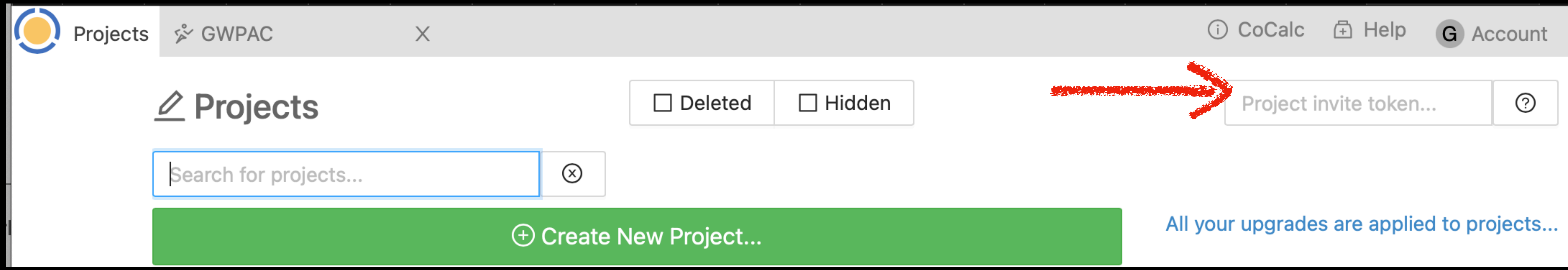
- First numerical calculation: computing π
- More python: numpy & matplotlib, functions
- Resolution, precision, accuracy
- Unix
- Parallel computing

- Open https://cocalc.com and sign in

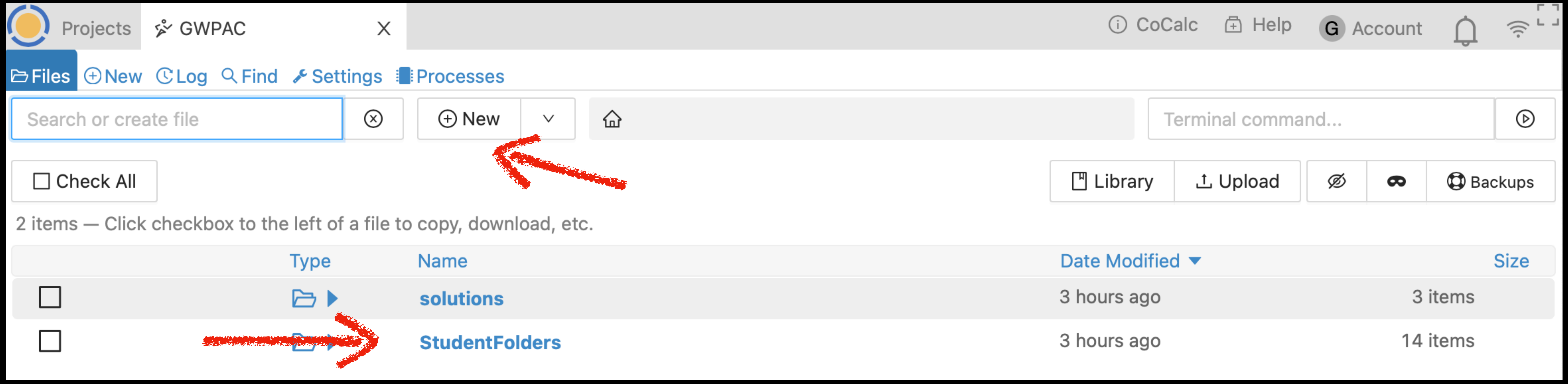


- See slack chat for the "token": enter it in the "token" box and press enter

  1UHueBCvBLsYj8Hz



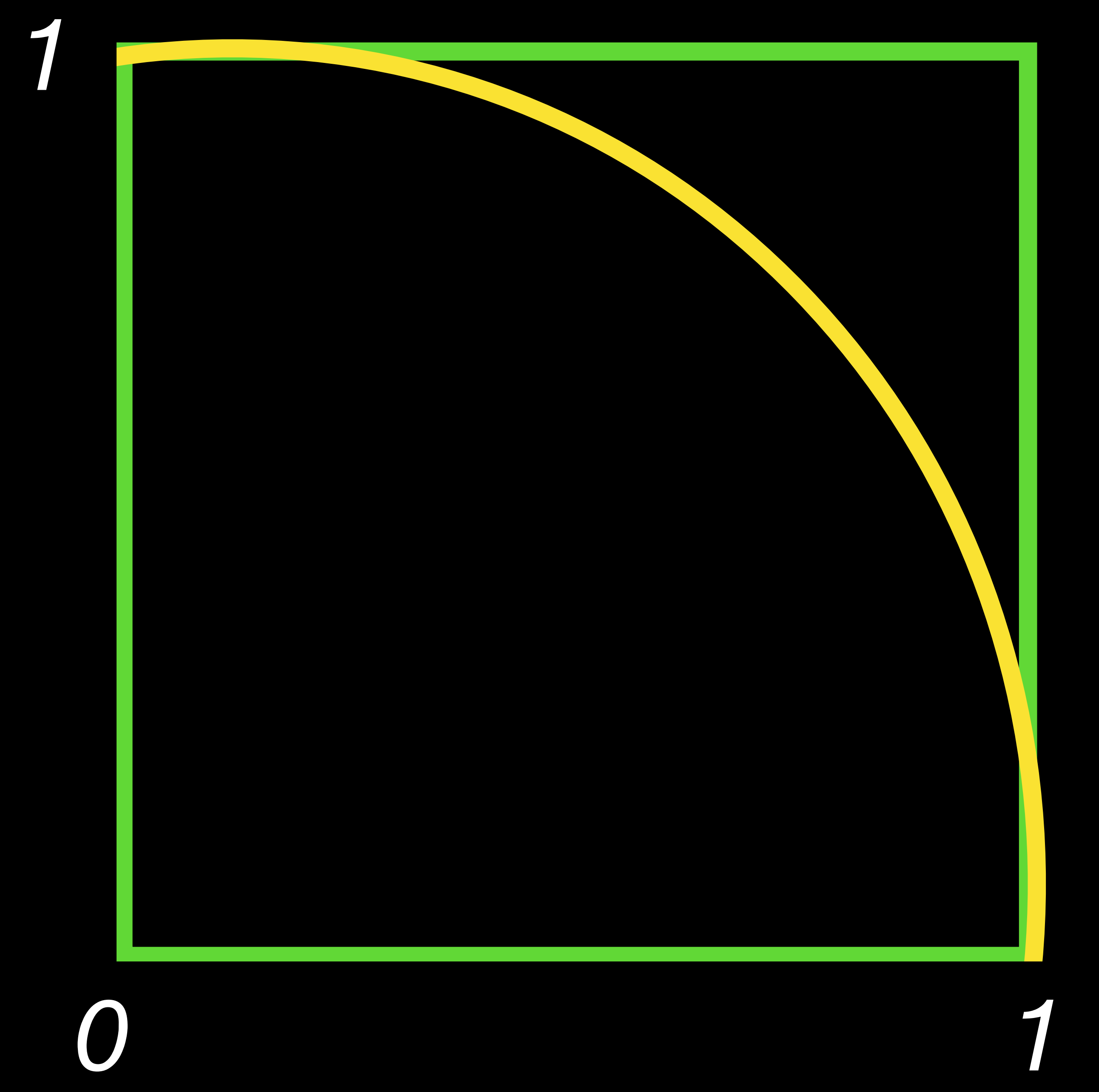
- Click "Day1.ipynb"



- Scroll to your name, and click in the box saying "`# Insert code here`" labeled below your name

  - Enter this code: VBBNN1tpwckqE5mw

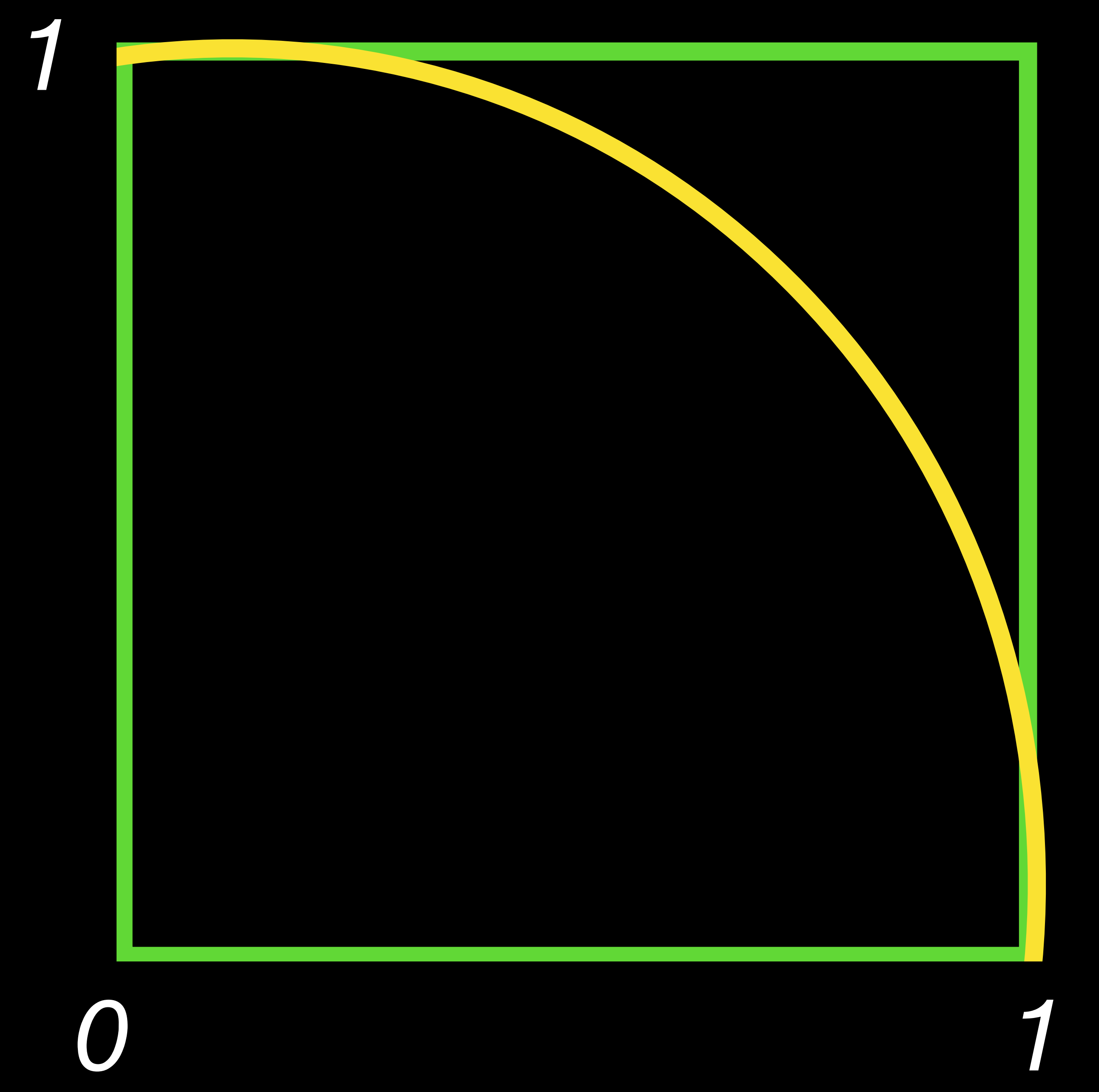
Pi Dartboard 3



- **Challenge:** Modify your program
  - Print $x^2 + y^2$ instead of just x and y

```
import math
import random

x = random.random()
y = random.random()

print(x)
print(y)
```

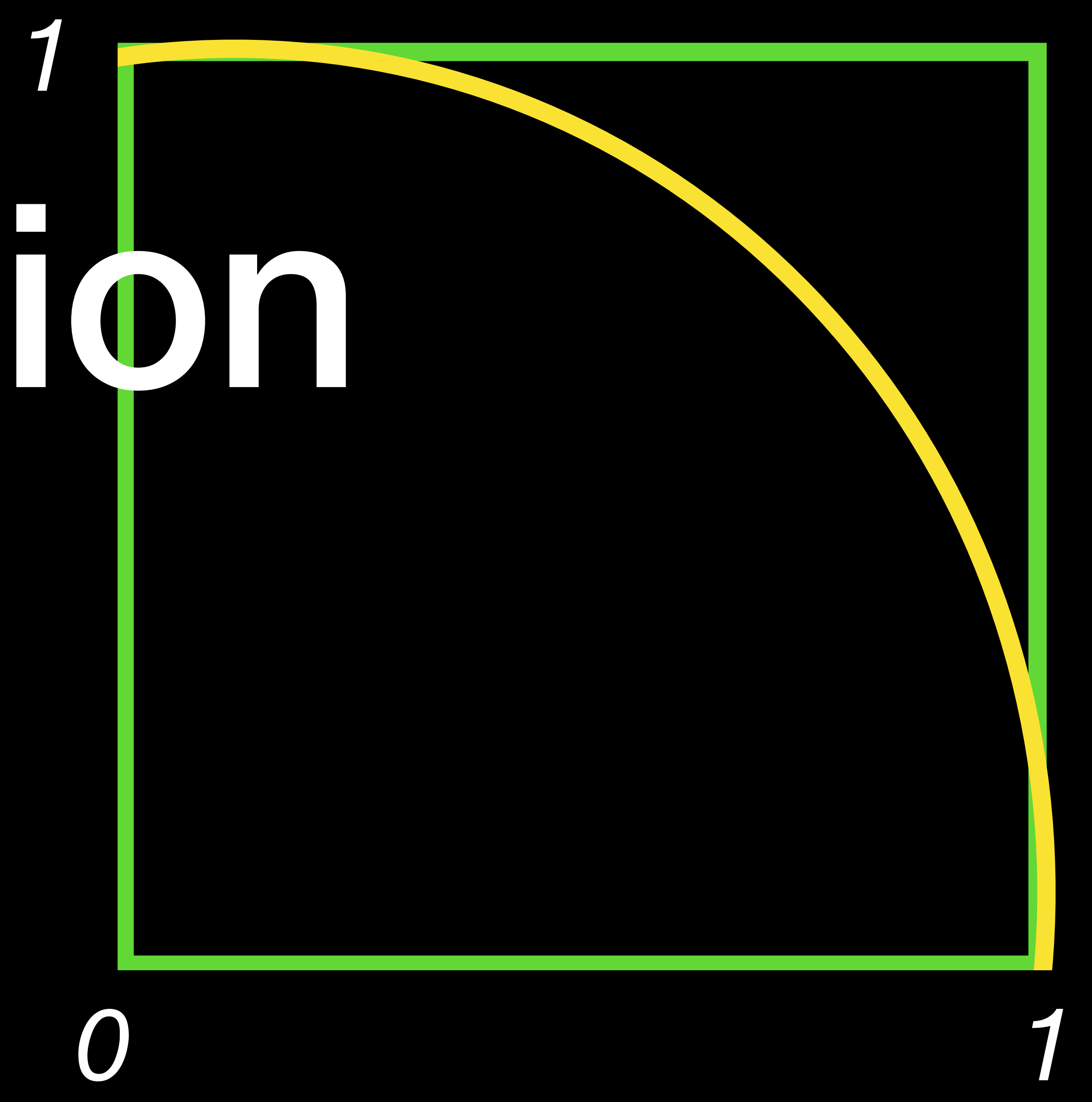
Pi Dartboard 3



- **Challenge:** Modify your program
  - Compute $x^2 + y^2$ and store it in a variable rSquared
  - Print rSquared instead of just x and y

```python
import math
import random

x = random.random()
y = random.random()

print(x)
print(y)
```

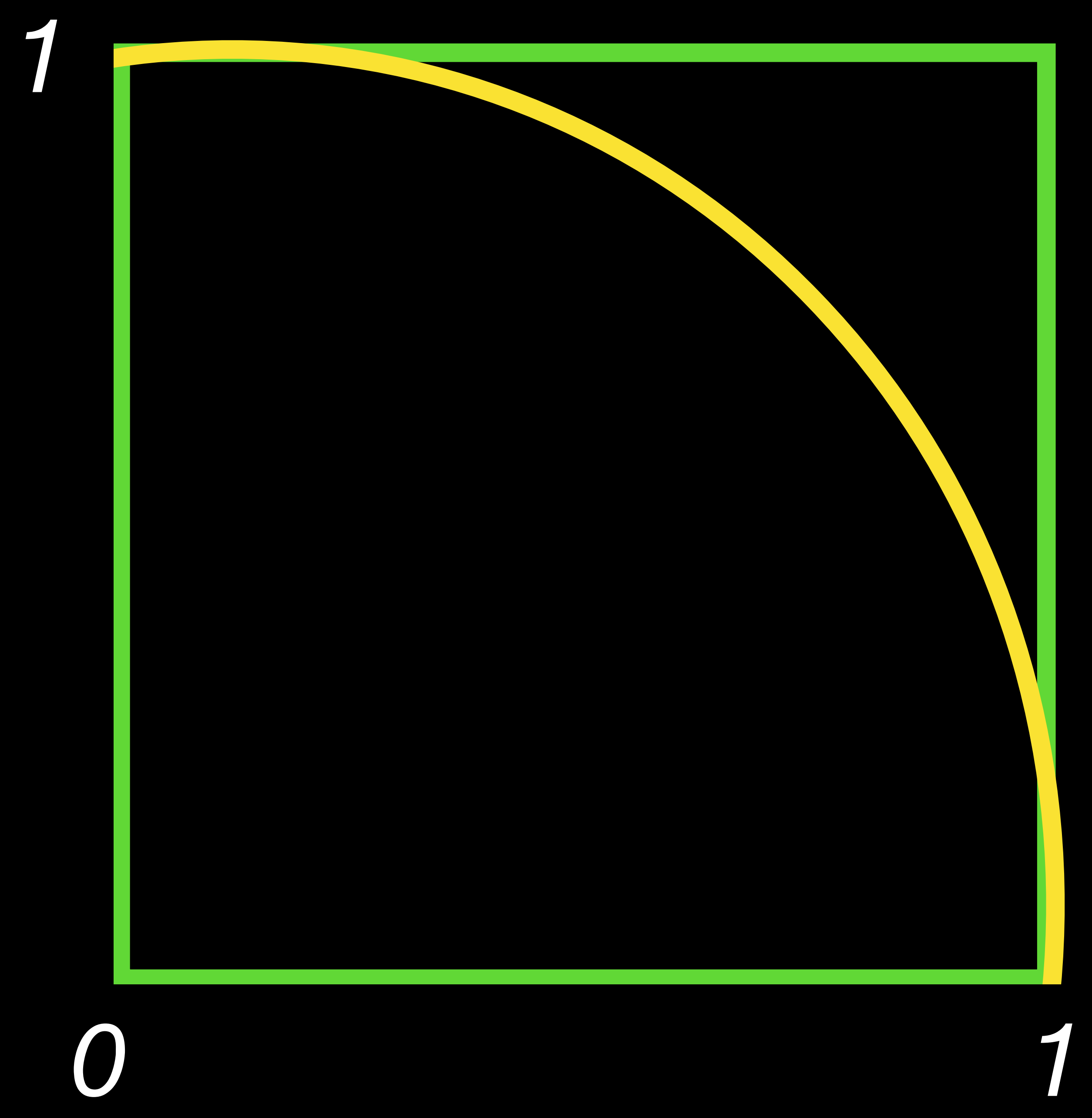
Pi Dartboard 3 Solution



- **Challenge:** Modify your program
  - Compute $x^2 + y^2$ and store it in a variable rSquared
  - Print rSquared instead of just x and y

```
import math
import random

x = random.random()
y = random.random()

rSquared = x**2 + y**2
print(rSquared)
```

Clicker question #2.5



- Which could be a number the program prints?

```
import math
import random

x = random.random()
y = random.random()

rSquared = x**2 + y**2
print(rSquared)
```

A -1.51

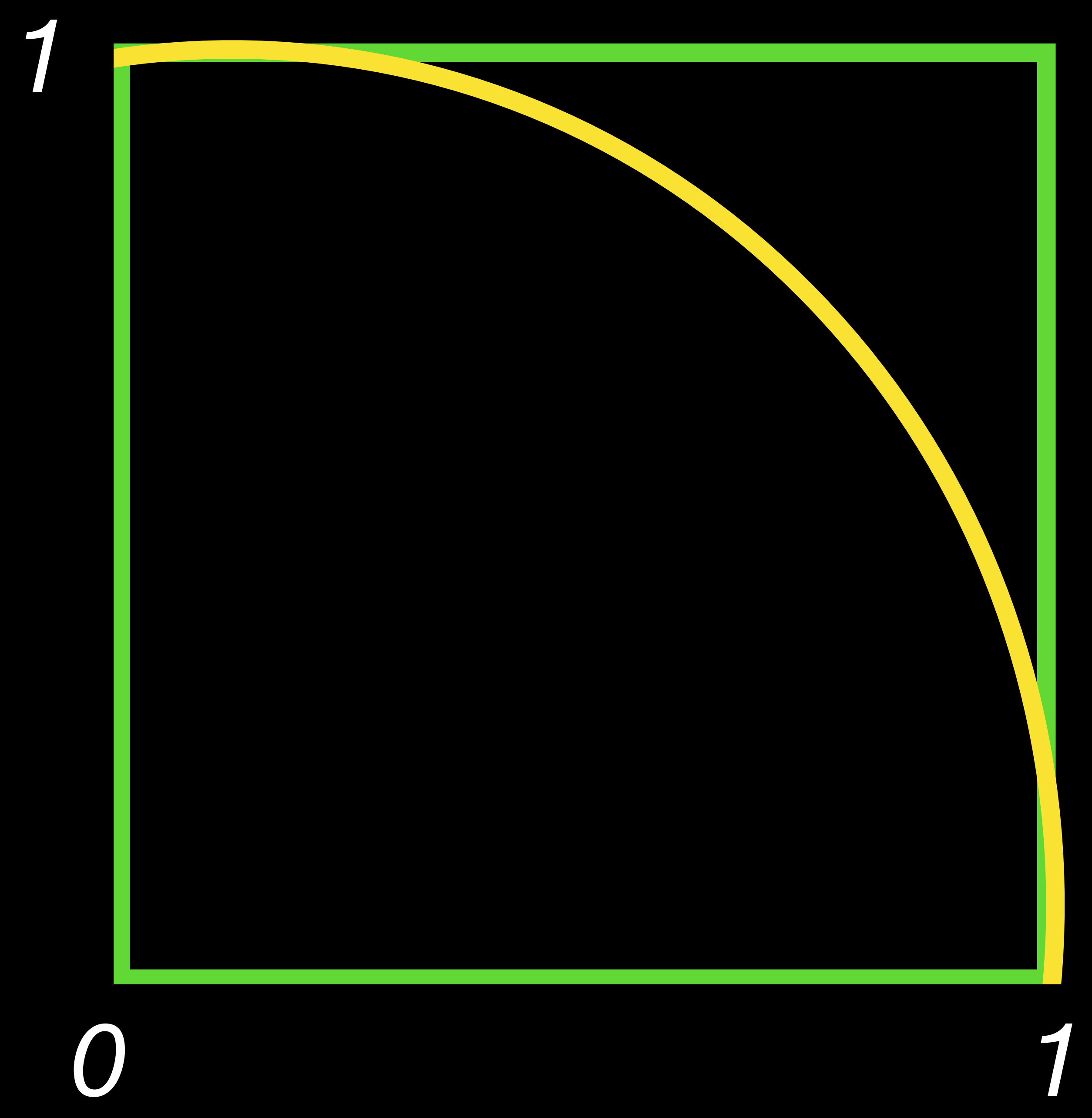
B 2.43

C -0.32

D 1.01

Clicker question #2.5

- If the dart is inside the **circle**, which could be the number printed by the program?




```
import math
import random

x = random.random()
y = random.random()

rSquared = x**2 + y**2
print(rSquared)
```

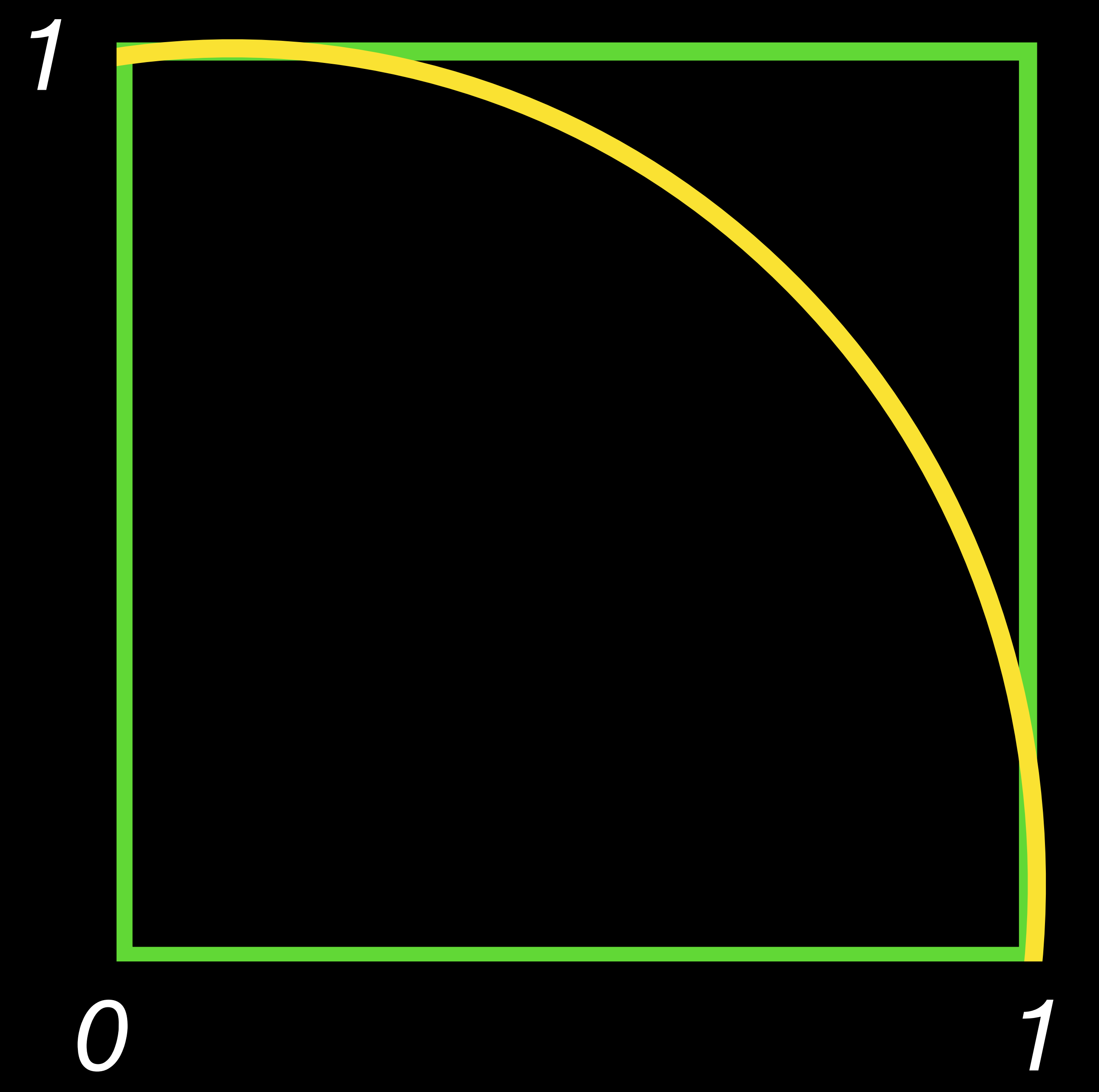
A 1.43

B 0.99

C 1.01

D More than one of ABC

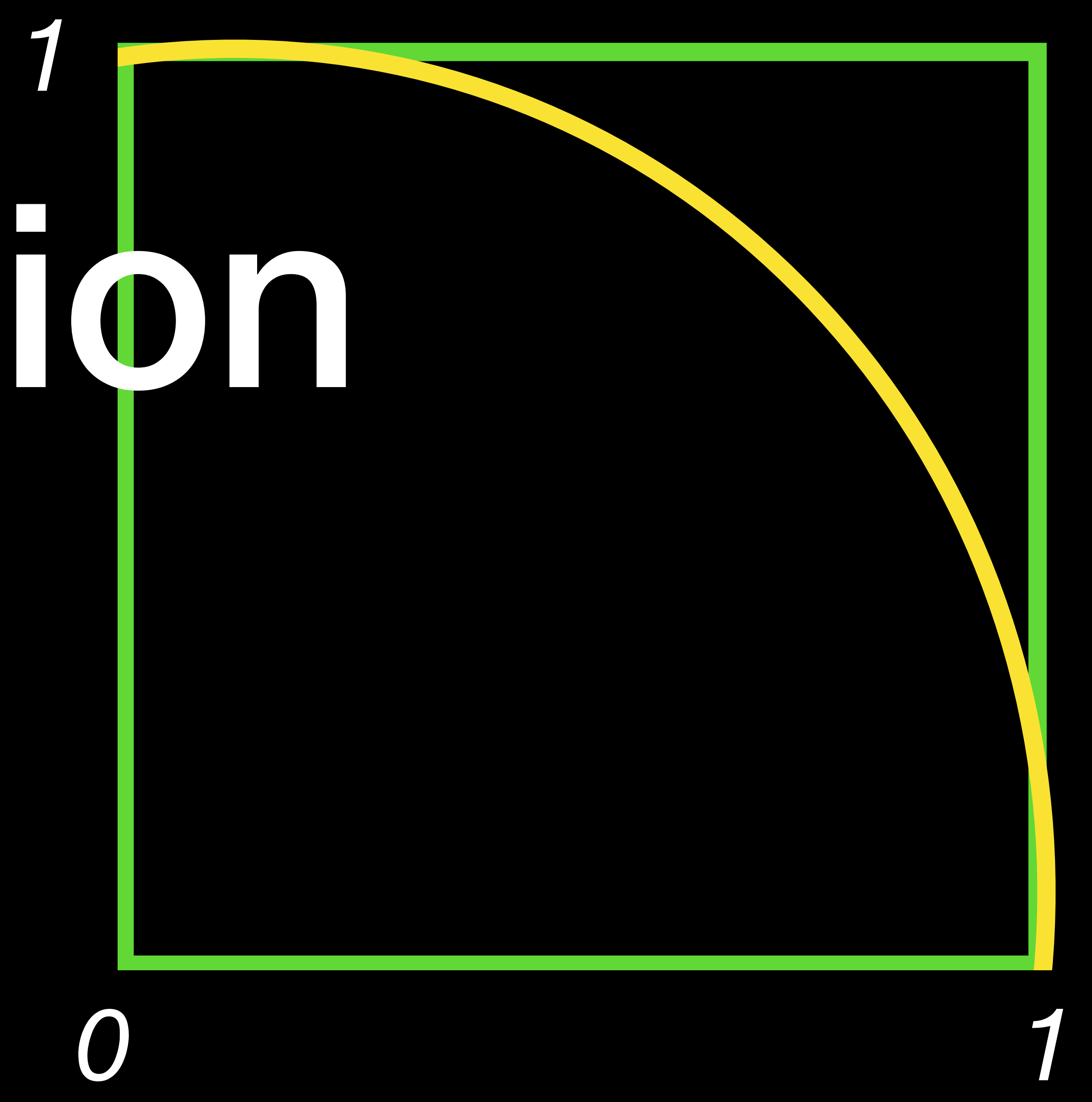
Pi Dartboard 4



- **Challenge:** Modify your program
  - Just below import random, make a new variable called “hits”, set to 0
  - If rSquared < 1, add 1 to hits
  - Print hits instead of rSquared

```
import math
import random
x = random.random()
y = random.random()
rSquared = x**2 + y**2
print(rSquared)
```

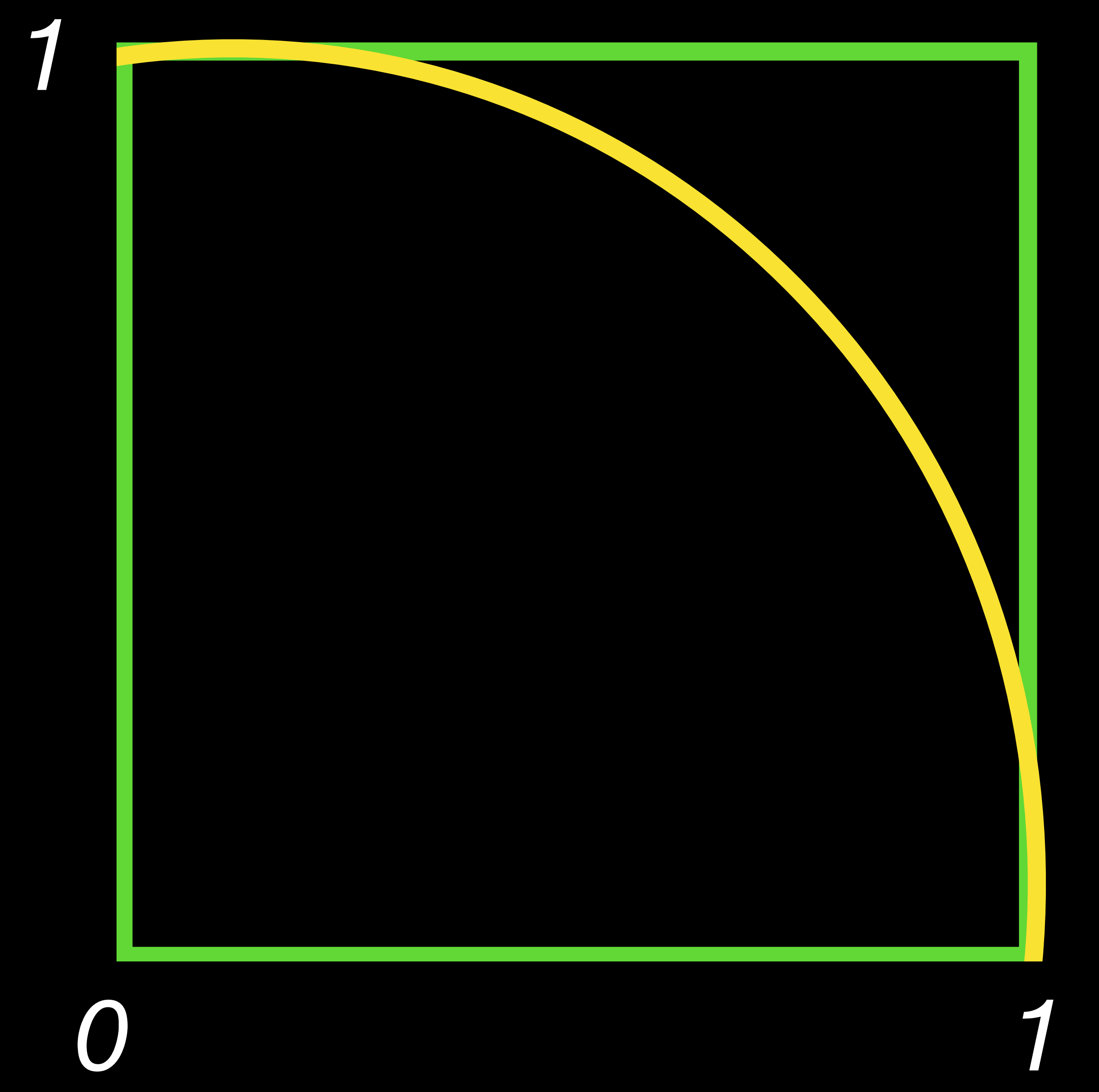
Pi Dartboard 4 Solution



- **Challenge:** Modify your program
  - Just below import random, make a new variable called “hits”, set to 0
  - If rSquared < 1, add 1 to hits
  - Print hits instead of rSquared

```
import math
import random
hits = 0
x = random.random()
y = random.random()
rSquared = x**2 + y**2
if rSquared < 1:
    hits = hits + 1
print(hits)
```

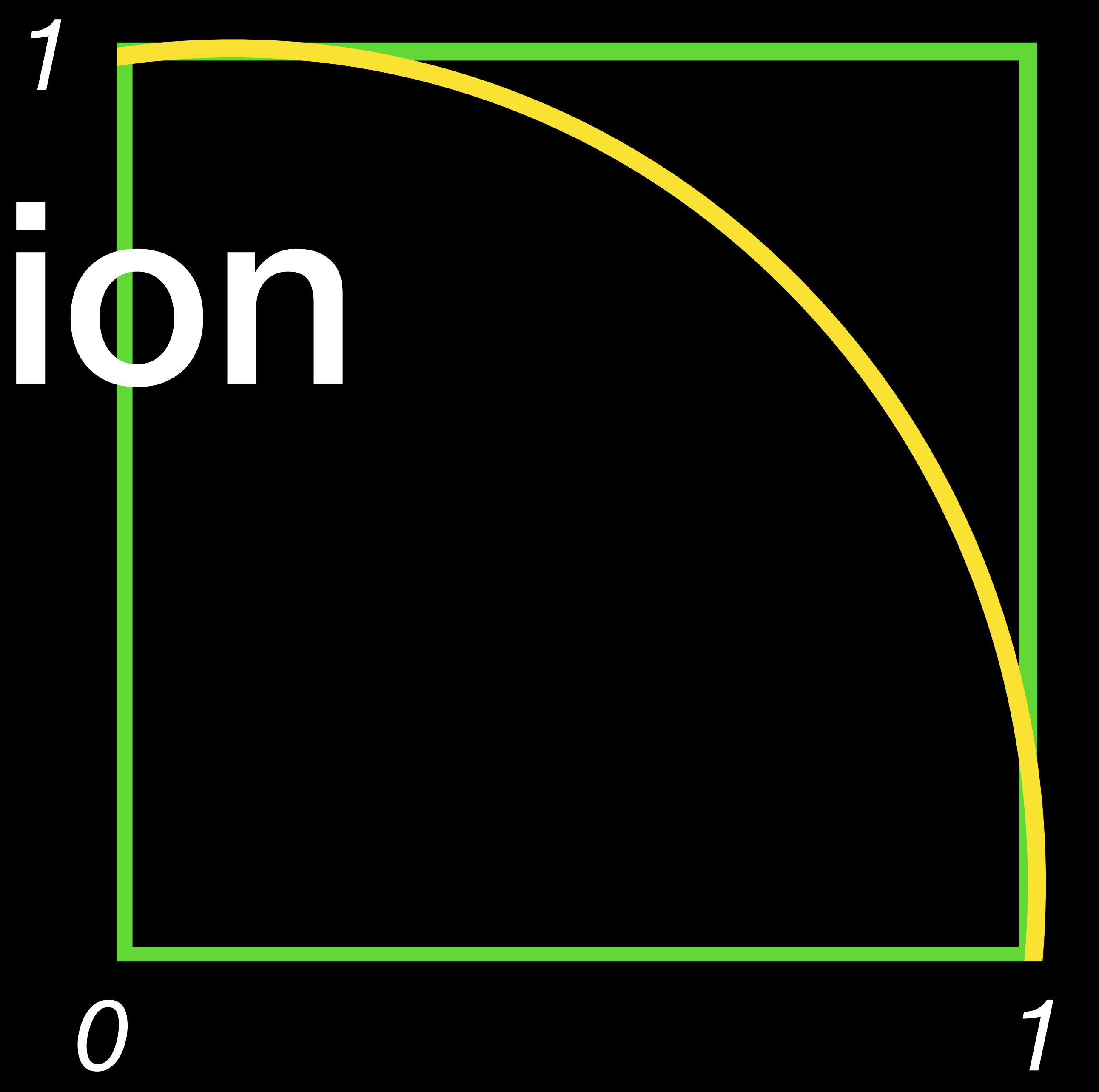
Pi Dartboard 5



- **Challenge:** Modify your program
  - Add a new variable, just below hits, called throws. Set it equal to 10.
  - Put the code that throws the dart and sees if it hit inside a while loop, so that you throw 10 darts instead of 1 dart
  - Don't forget to increment your while loop counter variable (i or j or whatever)

```
import math
import random
hits = 0
x = random.random()
y = random.random()
rSquared = x**2 + y**2
if rSquared < 1:
    hits = hits + 1
print(hits)
```

Pi Dartboard 5 Solution

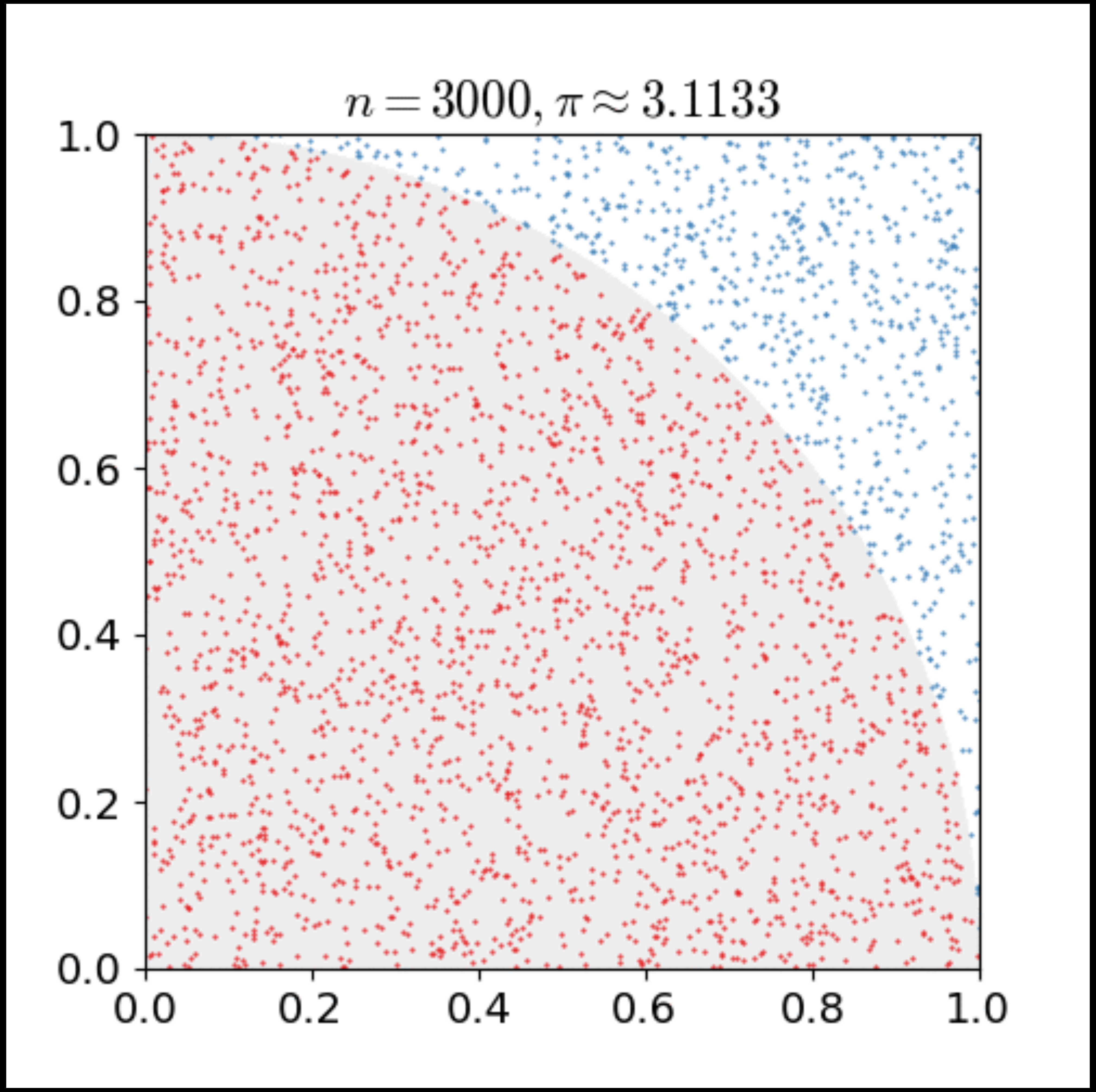


- **Challenge:** Modify your program
  - Add a new variable, just below hits, called throws. Set it equal to 10.
  - Put the code that throws the dart and sees if it hit inside a while loop, so that you throw 10 darts instead of 1 dart

```
import math
import random

hits = 0
throws = 10

i = 0
while i < throws:
    x = random.random()
    y = random.random()

    rSquared = x**2 + y**2
    if rSquared < 1:
        hits = hits + 1
    i = i + 1

print(hits)
```

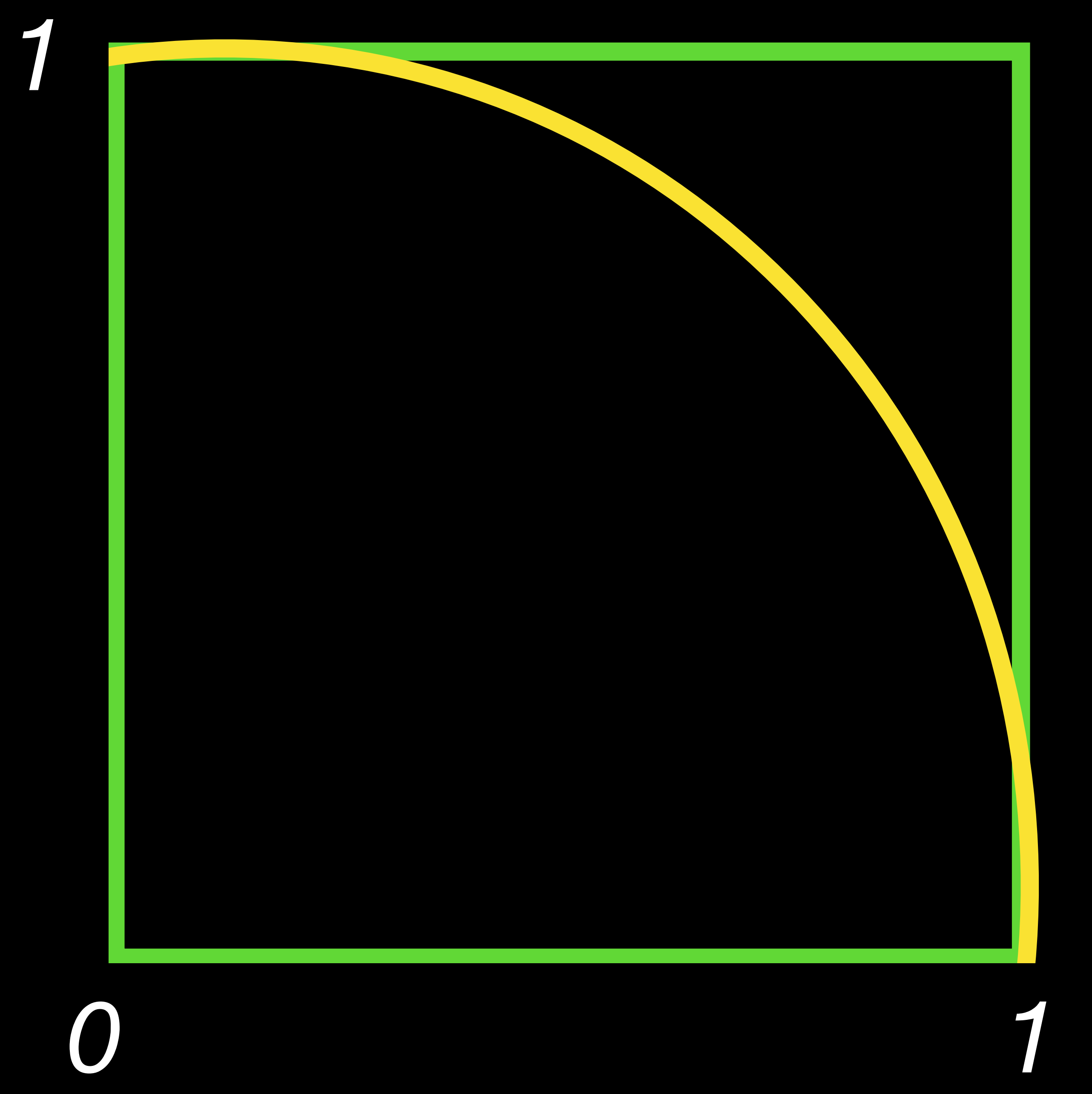
A silly way to compute π

- Throw darts in square
  - (circle area) ÷ (square area) ≈ hits ÷ throws = π/4
  - So π ≈ 4 * (hits ÷ throws)



Courtesy wikipedia

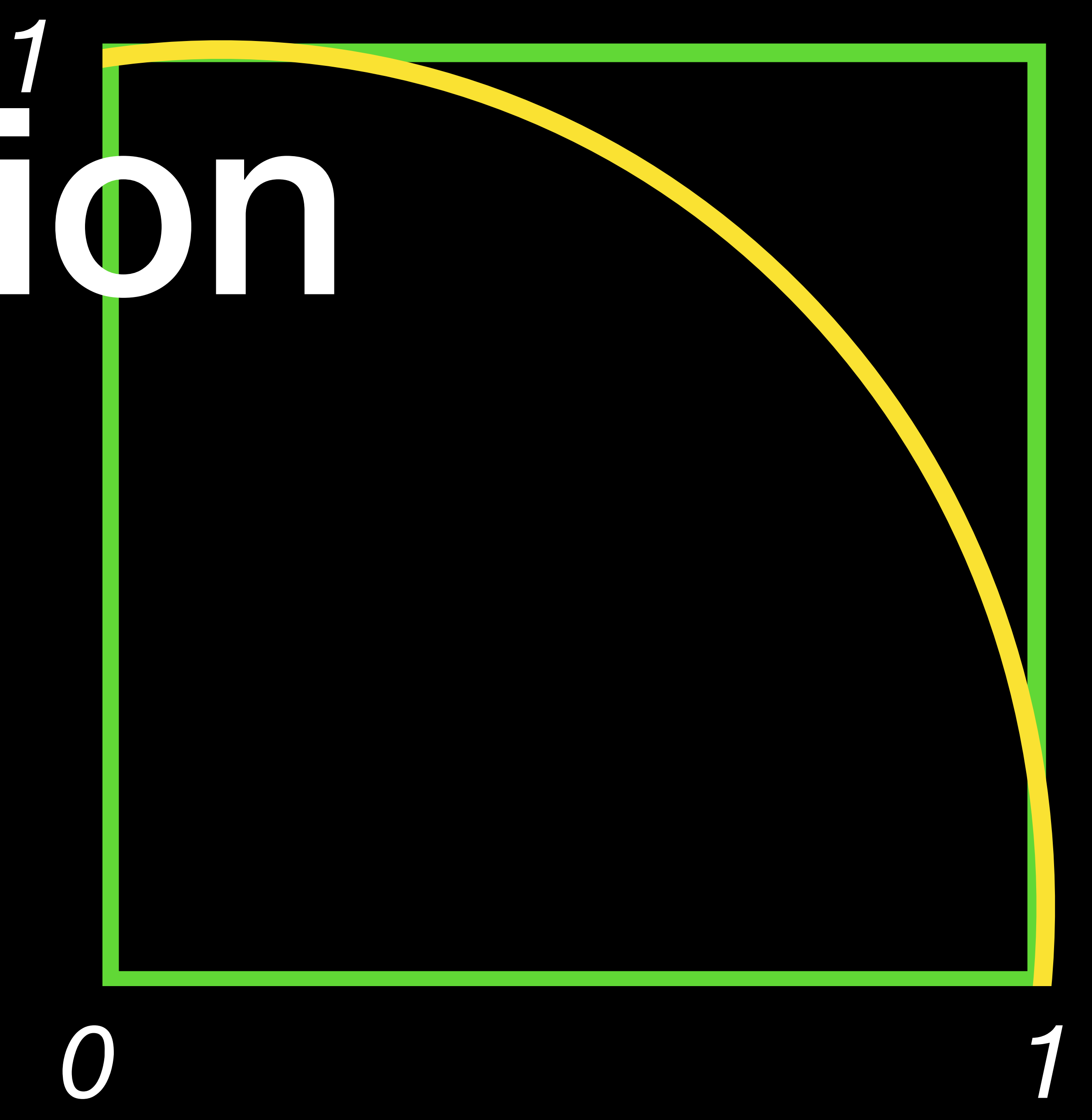
Pi Dartboard 6



- Finish the dartboard
  - Compute pi as 4.0 * float(hits) / float(throws)
  - Print your pi estimate

```
import math
import random
hits = 0
throws = 10
i = 0
while i < throws:
    x = random.random()
    y = random.random()

    rSquared = x**2 + y**2
    if rSquared < 1:
        hits = hits + 1
    i = i + 1
print(hits)
```

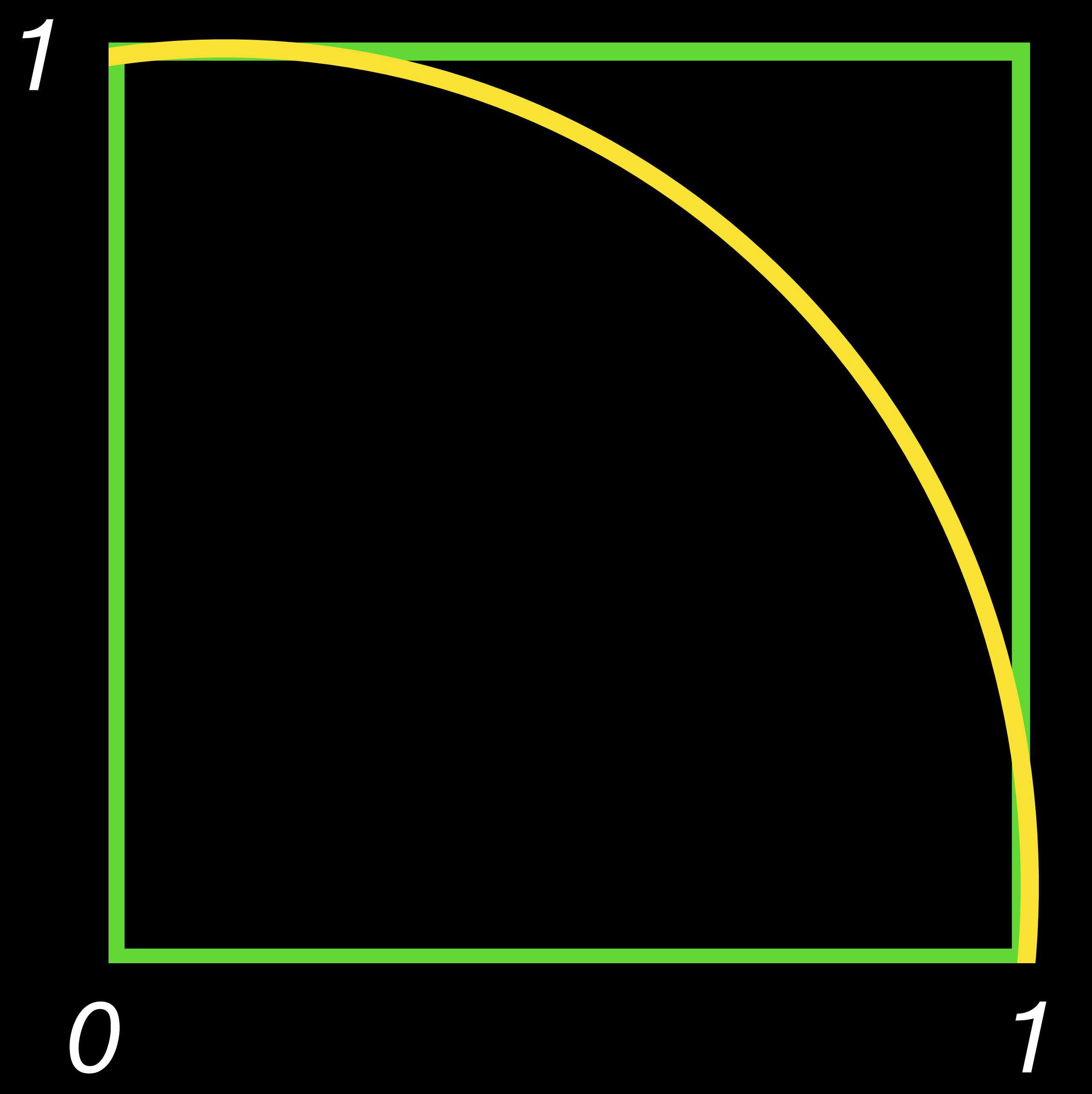
Pi Dartboard 6 Solution



- Finish the dartboard
  - Compute pi as 4.0 * float(hits) / float(throws)
  - Print your pi estimate

```
import math
import random

hits = 0
throws = 10

i = 0
while i < throws:
    x = random.random()
    y = random.random()

    rSquared = x**2 + y**2
    if rSquared < 1:
        hits = hits + 1
    i = i + 1

pi = 4.0 * float(hits) / float(throws)
print(pi)
```

Pi Dartboard 7



- See what happens as you make throws 10**n, n=1,2,3,4,5,6,7

- For n=7, how does speed compare if you do `rSquared = x*x + y*y`

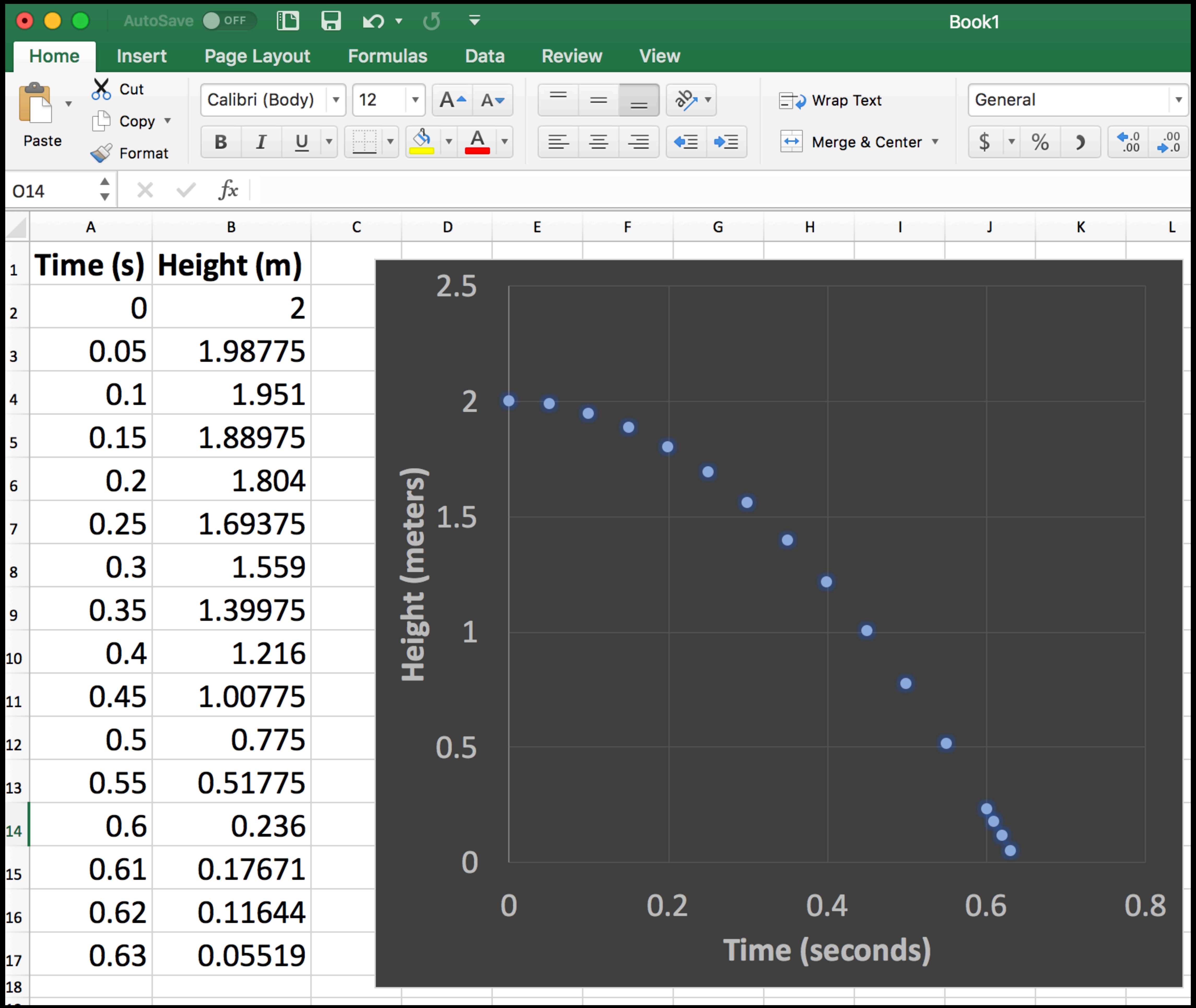
```
import math
import random

hits = 0
throws = 10

i = 0
while i < throws:
    x = random.random()
    y = random.random()

    rSquared = x**2 + y**2
    if rSquared < 1:
        hits = hits + 1
    i = i + 1

pi = 4.0 * float(hits) / float(throws)
print(pi)
```

Plotting your results

- Scatter plots
- Lists and numpy arrays
- Pyplot plotting

Scatter plots

- Data: result or output given some input
- Example: dropped marker height vs. time
- Tools to make scatter plots
  - Excel
  - Python
    - Lists of numbers
    - Computations on lists of numbers: numpy arrays
    - pyplot: makes scatter plots



Lists

- **Values -** ordered sets of objects, all the same type (like floats or ints)
- **Operators -** **[ ], .append()**
- **Easily add on elements in loops** with .append()

```
x = [-2.0, -1.0, 0.0, 1.0, 2.0]
y = ["Hello", "world"]
z = [1, 4, 9, 16]
```

```
z.append(25) == [1, 4, 9, 16, 25]
```

```
x[0] == -2.0
x[1] == -1.0
x[4] == 2.0
y[0] == "Hello"
y[-1] == "world"
z[-1] == 16
z[0] == 1
```

Loop over lists

```
for i in [0,1,2,3]:
  print(i*i)
```

```
0
1
4
9
```

```
import numpy as np
print(np.arange(1,5,1))
[1,2,3,4]
```

```
import numpy as np
myCountArray = np.arange(1,5)
myList = []
for i in myCountArray:
    myList.append(i*i)
print(myCountArray)
print(myList)
```

```
[1,2,3,4]
[1,4,9,16]
```

Clicker question #2.6

- What value does the program print?

```
x = [1.0, 4.0, 9.0]
print(x[1])
```

A	1.0
B	4.0
C	9.0
D	The entire list

Numpy arrays

- **Values -** ordered sets of objects, all the same type (like floats or ints)
- **Operators - [ ], +, -, *, /, np.sqrt(), np.sin(), np.cos(), …**
- **Easily do math on whole lists at once** (like formulas in Excel)

```
x = np.array([-2.0, -1.0, 0.0, 1.0, 2.0])
y = np.array(["Hello", "world"])
q = np.array([1, 2, 3, 4])
r = q * 2
s = q + r
z = q * q
```

```
r == np.array([2, 4, 6, 8])
```

```
s == np.array([3, 6, 9, 12])
z == np.array([1, 4, 9, 16])
```

```
x[0] == -2.0
x[1] == -1.0
x[4] == 2.0
y[0] == "Hello"
y[-1] == "world"
z[-1] == 16
z[0] == 1
```

Making sample data

- Annoying to type [1,2,3,4,…] all the time
- Instead: np.arange(start, stop, step)
- What do all these numbers mean??
  - Make a plot to visualize them

Try in cocalc!

```
import numpy as np
x = np.arange(-4.0, 4.0, 0.01)
y = np.sin(x)**3
print(x)
print(y)
```

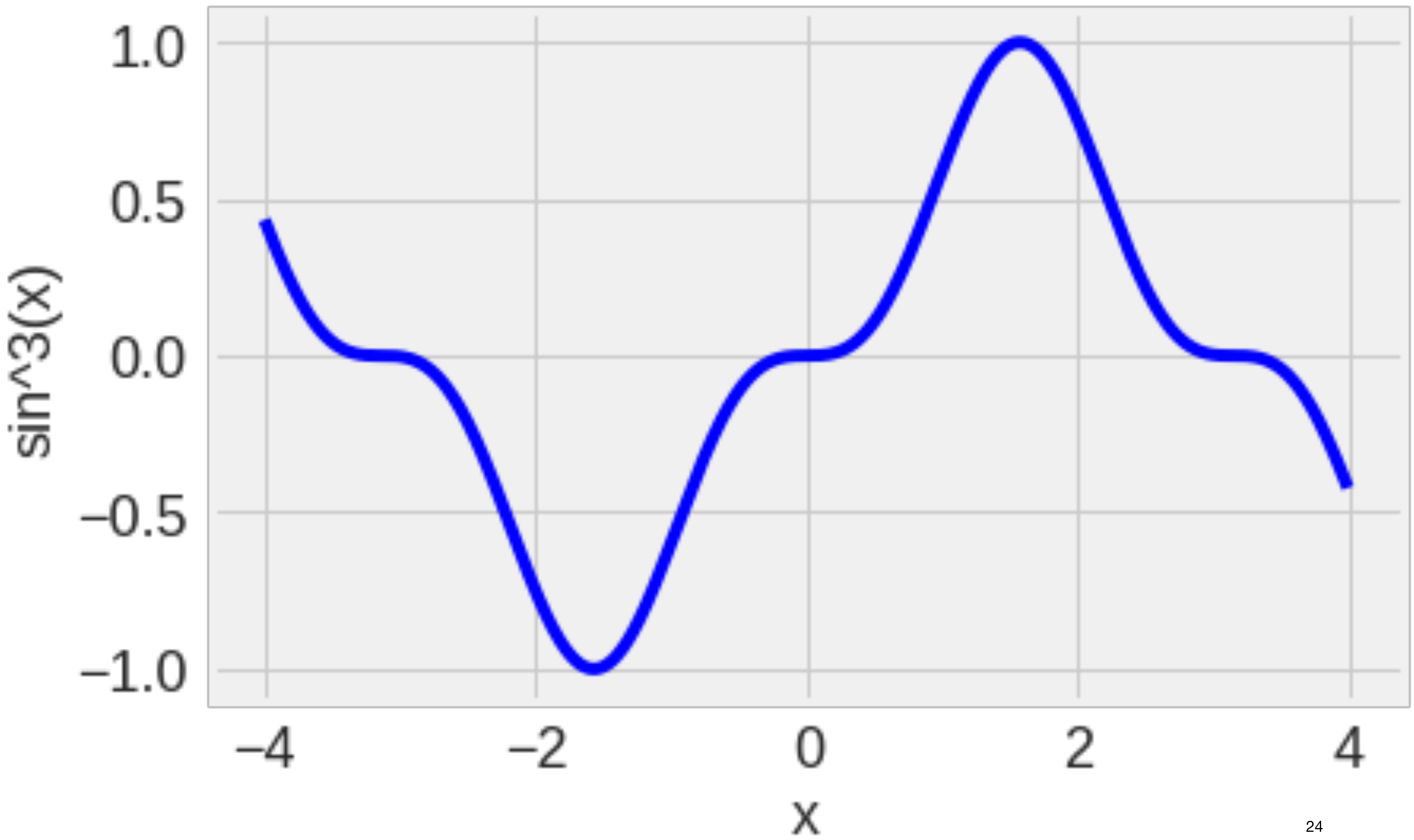
```
-9.99825171e-01 -9.99351433e-01 -9.98578166e-01 -9.97505912e-01
-9.96135421e-01 -9.94467651e-01 -9.92503769e-01 -9.90245148e-01
-9.87693366e-01 -9.84850205e-01 -9.81717651e-01 -9.78297888e-01
-9.74593301e-01 -9.70606471e-01 -9.66340175e-01 -9.61797379e-01
-9.56981241e-01 -9.51895105e-01 -9.46542499e-01 -9.40927131e-01
-9.35052889e-01 -9.28923832e-01 -9.22544191e-01 -9.15918365e-01
-9.09050915e-01 -9.01946561e-01 -8.94610179e-01 -8.87046794e-01
-8.79261581e-01 -8.71259853e-01 -8.63047062e-01 -8.54628794e-01
-8.46010761e-01 -8.37198799e-01 -8.28198860e-01 -8.19017011e-01
-8.09659425e-01 -8.00132377e-01 -7.90442239e-01 -7.80595473e-01
-7.70598629e-01 -7.60458333e-01 -7.50181290e-01 -7.39774268e-01
-7.29244102e-01 -7.18597680e-01 -7.07841944e-01 -6.96983877e-01
-6.86030504e-01 -6.74988880e-01 -6.63866088e-01 -6.52669231e-01
-6.41405427e-01 -6.30081800e-01 -6.18705479e-01 -6.07283586e-01
-5.95823237e-01 -5.84331527e-01 -5.72815532e-01 -5.61282298e-01
-5.49738839e-01 -5.38192126e-01 -5.26649084e-01 -5.15116589e-01
-5.03601455e-01 -4.92110435e-01 -4.80650212e-01 -4.69227393e-01
-4.57848505e-01 -4.46519990e-01 -4.35248195e-01 -4.24039375e-01
-4.12899678e-01 -4.01835147e-01 -3.90851715e-01 -3.79955193e-01
-3.69151273e-01 -3.58445520e-01 -3.47843366e-01 -3.37350109e-01
-3.26970907e-01 -3.16710771e-01 -3.06574566e-01 -2.96567003e-01
-2.86692639e-01 -2.76955868e-01 -2.67360924e-01 -2.57911871e-01
```

Plotting sample data

- Make plots with pyplot

Try in cocalc!

```
import numpy as np
import matplotlib
from matplotlib import pyplot as plt
matplotlib.rc('axes', labelsize=18)
matplotlib.rc('xtick', labelsize = 18)
matplotlib.rc('ytick', labelsize = 18)

x = np.arange(-4.0, 4.0, 0.01)
y = np.sin(x)**3

plt.clf() #clear figure
plt.plot(x,y, color='b')
plt.xlabel('x')
plt.ylabel('sin^3(x)')
plt.show()
```


1.0
0.5
0.0
−0.5
−1.0
sin^3(x)
−4
−2
0
2
4
x

Functions

- Input(s) ("arguments")
- Returns output
- Functions can call other functions

Try in cocalc!

```
def square(x):
  return x**x
```

```
square(4)
16
```

Plotting π 1

- **Activity: edit your code**
  - Make everything but the last two lines inside a function that takes one input, n
  - Instead of setting throws = 10, set throws=n
  - Return the pi estimate

```
import math
import random

hits = 0
throws = 10

i = 0
while i < throws:
    x = random.random()
    y = random.random()

    rSquared = x*x + y*y
    if rSquared < 1:
        hits = hits + 1
    i = i + 1
pi = 4.0 * float(hits) / float(throws)
print(pi)
```

Solution 1

- **Activity: edit your code**
  - Make everything but the last two lines inside a function that takes one input, n
  - Instead of setting throws = 10, set throws=n
  - Return the pi estimate

```
import math
import random

def estimatePi(throws):
  hits = 0
  i = 0
  while i < throws:
        x = random.random()
        y = random.random()

        rSquared = x*x + y*y
        if rSquared < 1:
              hits = hits + 1
        i = i + 1

  pi = 4.0 * float(hits) / float(throws)
  return pi

print(estimatePi(1e4))
```

Plotting π 2

```python
import math
import random

def estimatePi(throws):
  # (same definition of estimatePi function here)
  return pi

piEstimates = [estimatePi(x) for x in [10, 100, 1000, 10000]]
print(piEstiamtes)
```

Plotting π 2

```
import math
import random

def estimatePi(throws):
  # (same definition of estimatePi function here)
  return pi

trials = [10**j for j in np.arange(0,6)]
piEstimates = [estimatePi(x) for x in trials]
print(trials, piEstimates)
```

Plotting π 2

```
import math
import random

def estimatePi(throws):
  # (same definition of estimatePi function here)
  return pi

trials = [10**j for j in np.arange(0,6)]
piEstimates = [estimatePi(x) for x in trials]
print(trials, piEstimates)

plt.clf()
plt.plot(x,y)
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

Plotting π 2

```python
import math
import random

def estimatePi(throws):
  # (same definition of estimatePi function here)
  return pi

trials = [10**j for j in np.arange(0,6)]
piEstimates = [estimatePi(x) for x in trials]
print(trials, piEstimates)

plt.clf()
plt.plot(trials,piEstimates)
plt.xlabel("Darts thrown")
plt.ylabel("pi Estimate")
plt.xscale('log')
plt.show()
```

Plotting π 2

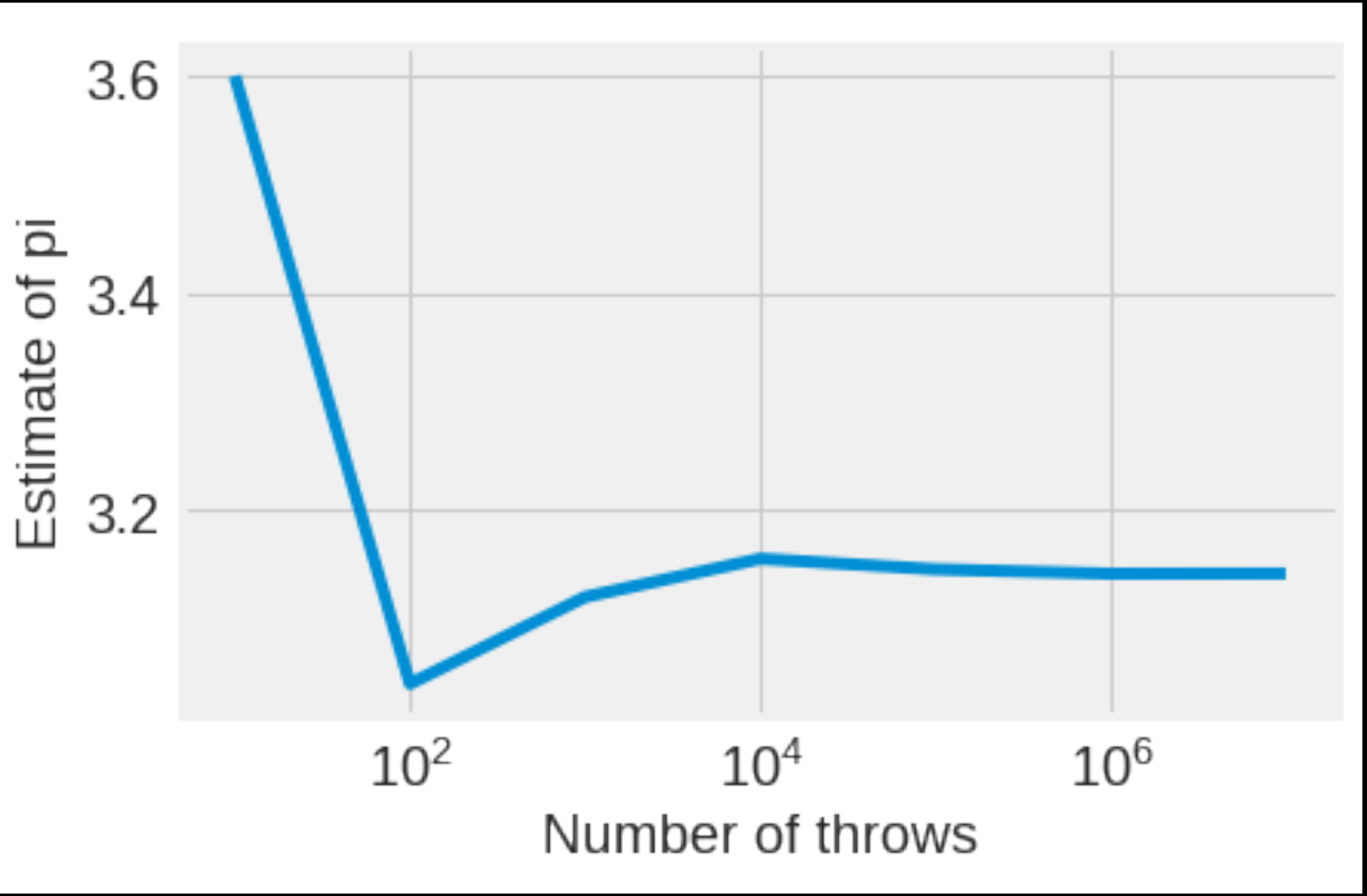
```python
import math
import random

def estimatePi(throws):
  # (same definition of estimatePi function here)
  return pi

trials = [10**j for j in np.arange(0,6)]
piEstimates = [estimatePi(x) for x in trials]
print(trials, piEstimates)

plt.clf()
plt.plot(trials,np.abs(np.array(piEstimates)-np.pi))
plt.xlabel("Darts thrown")
plt.ylabel("pi Estimate")
plt.xscale('log')
plt.show()
```

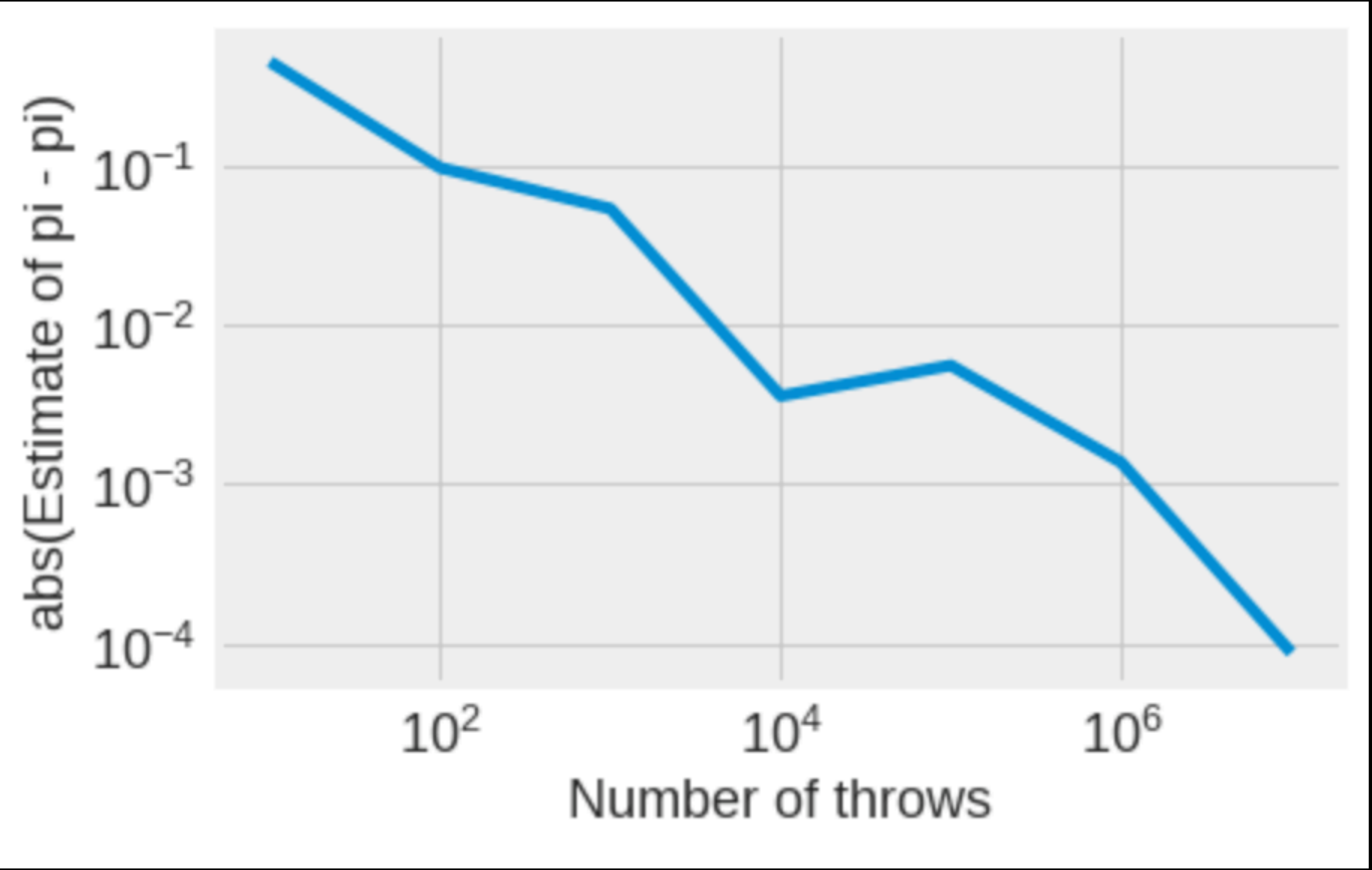
Plotting π 2

```
import math
import random

def estimatePi(throws):
  # (same definition of estimatePi function here)
  return pi

trials = [10**j for j in np.arange(0,6)]
piEstimates = [estimatePi(x) for x in trials]
print(trials, piEstimates)

plt.clf()
plt.plot(trials,np.abs(np.array(piEstimates)-np.pi))
plt.xlabel("Darts thrown")
plt.ylabel("pi Estimate")
plt.xscale('log')
plt.yscale('log')
plt.show()
```

Accuracy of the π dart board

- As throws goes up, answer gets closer to pi
- But it's hard to see how close it is later on
- So instead, plot the difference between the estimate and the real answer





abs(Estimate of pi - pi)
10⁻¹
10⁻²
10⁻³
10⁻⁴
10²
10⁴
10⁶
Number of throws

Concepts in numerical programming

- Resolution
- Accuracy
- Precision

Resolution

Low resolution

Entire image: 227KB

Large galaxies
1 billion light years away

Small galaxies up to
13 billion light years away

Image courtesy NASA

Resolution

High resolution

Entire image: 110MB

Large galaxies
1 billion light years away

Small galaxies up to
13 billion light years away

Image courtesy NASA

Resolution

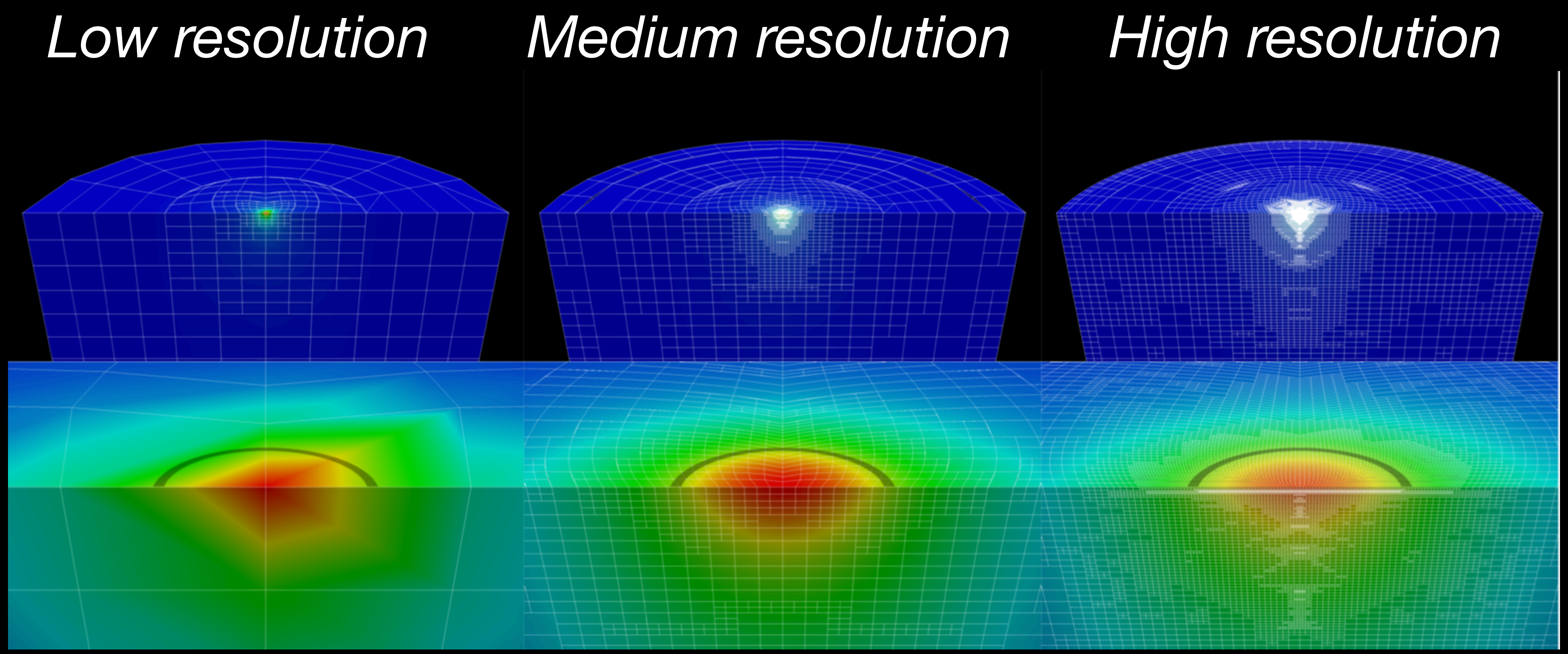
- **Low resolution**
  - Smaller data
  - Faster computation
  - Less precise
- **High resolution**
  - Bigger data
  - Slower computation
  - More precise

Precision & accuracy

- **Precision**
  - How much result changes when you add more resolution
  - "How many digits"
- **Accuracy**
  - How close result is to the correct result

Example: thermal noise

- Thermal noise of a mirror in LIGO depends on how much potential energy it gets when you push on the face

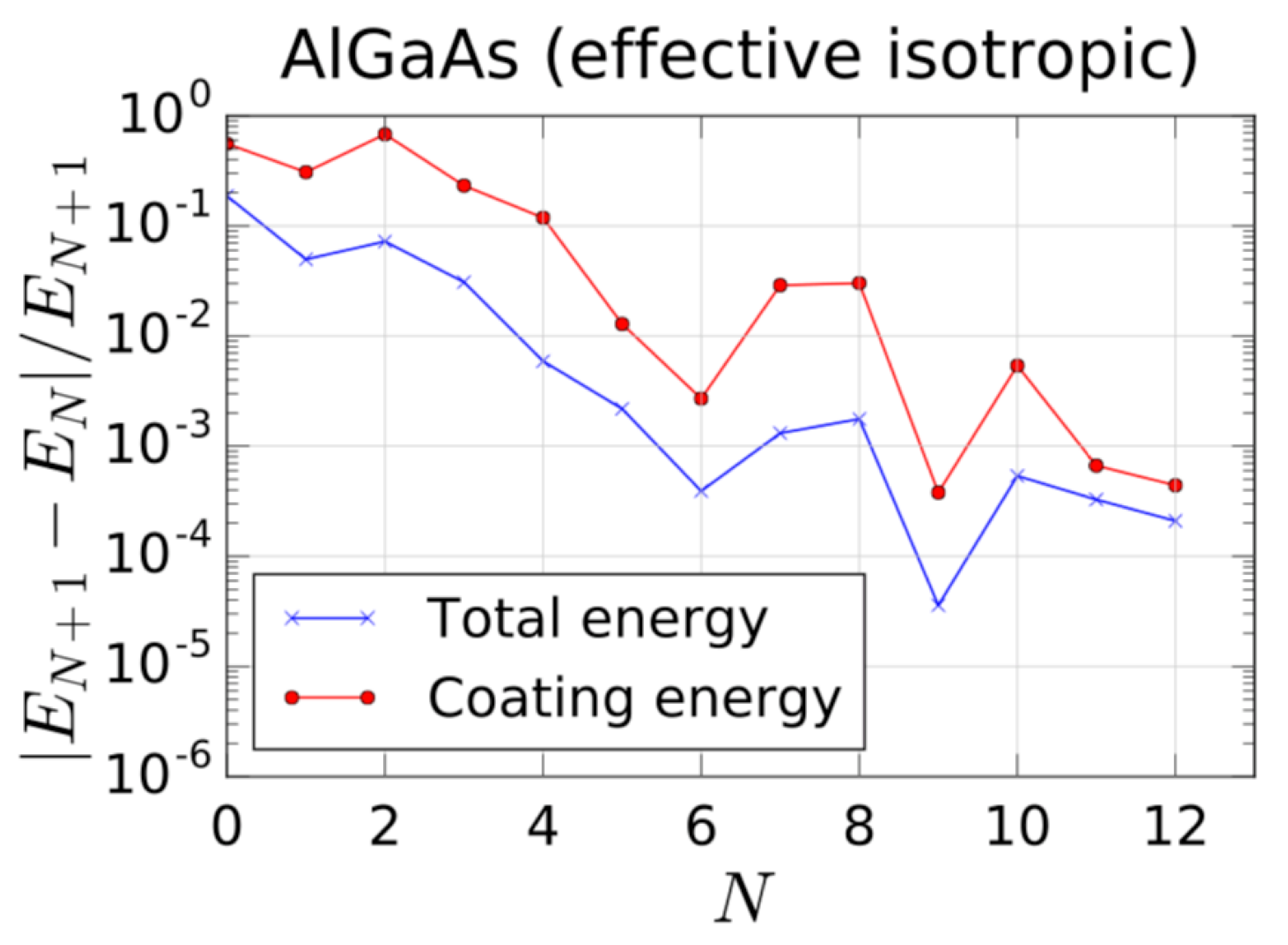


Color = how much mirror deforms

Lovelace, Demos, Khan (2018)

Example: thermal noise

- Potential energy *E* in deformed mirror
  - Precision of energy as resolution increases
    - Label resolution by integer *N*

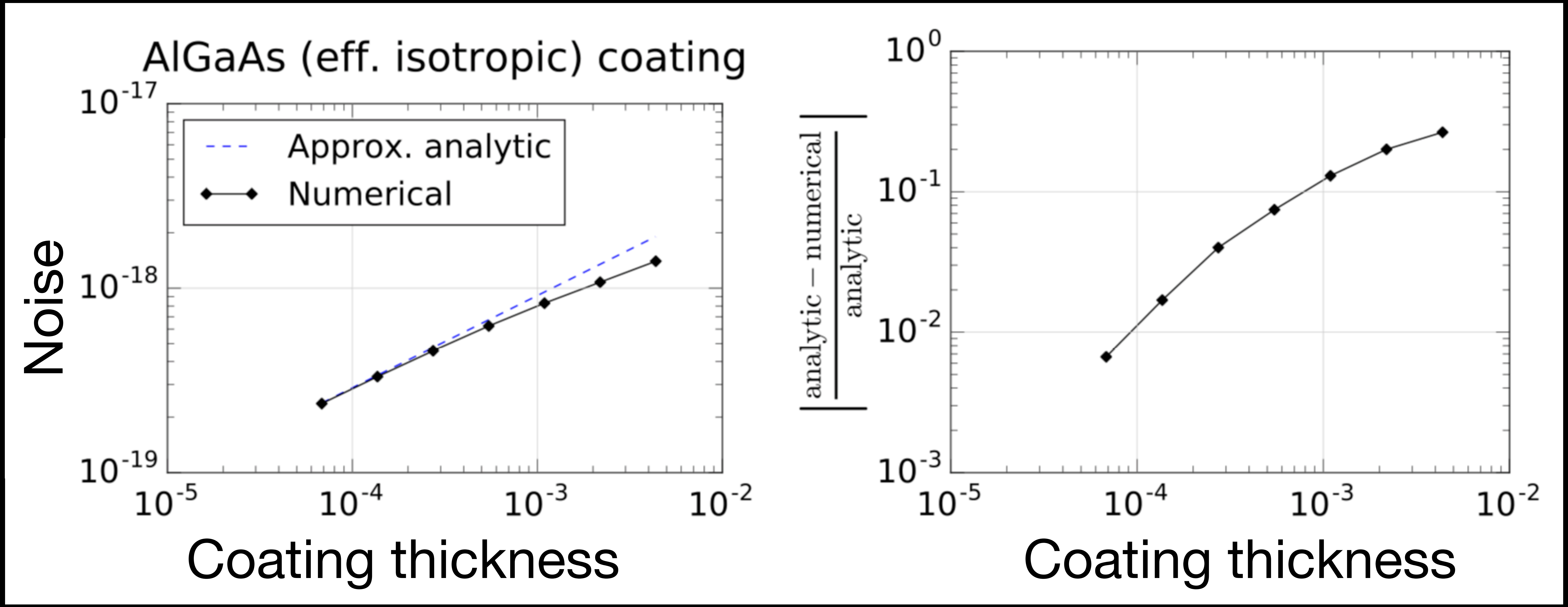


Higher resolution

Lovelace, Demos, Khan (2018)

Example: thermal noise

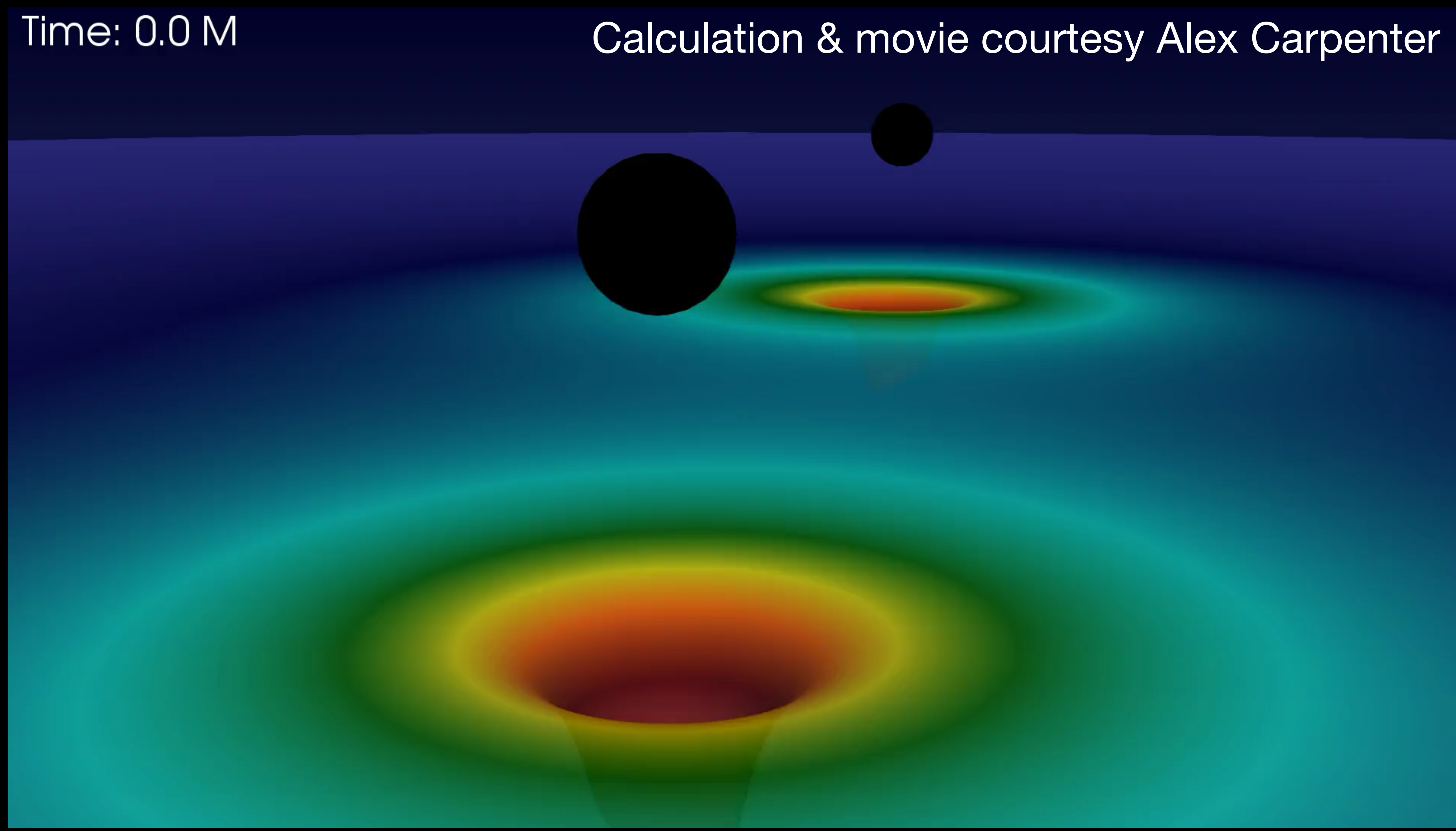
- Thermal noise of thin coating
  - Accuracy: compare code to known "analytic" solution



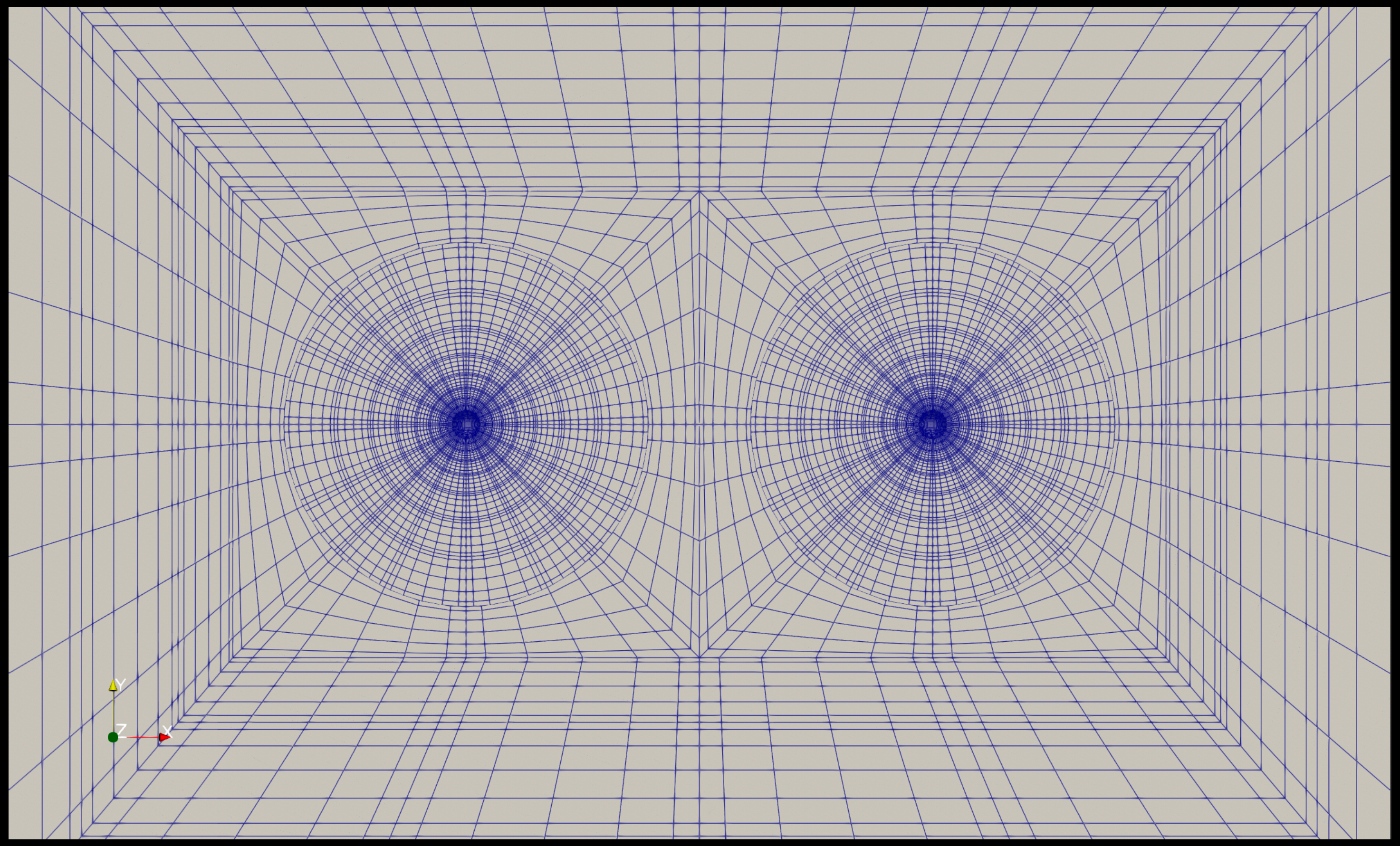
Lovelace, Demos, Khan (2018)

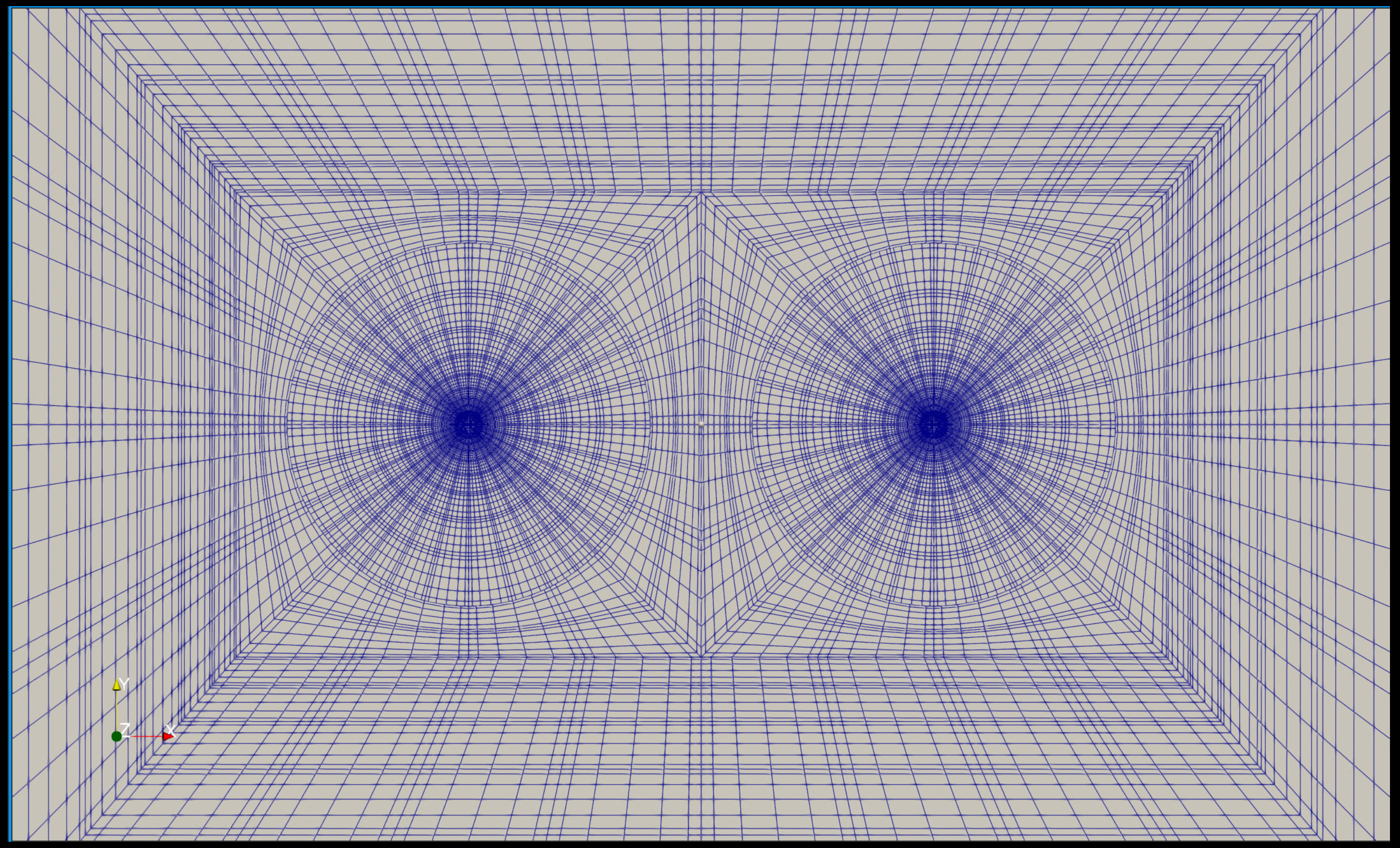
Example of resolution

3-index constraint



2.5k elements
7^3 points/element





13k elements
7^3 points/element