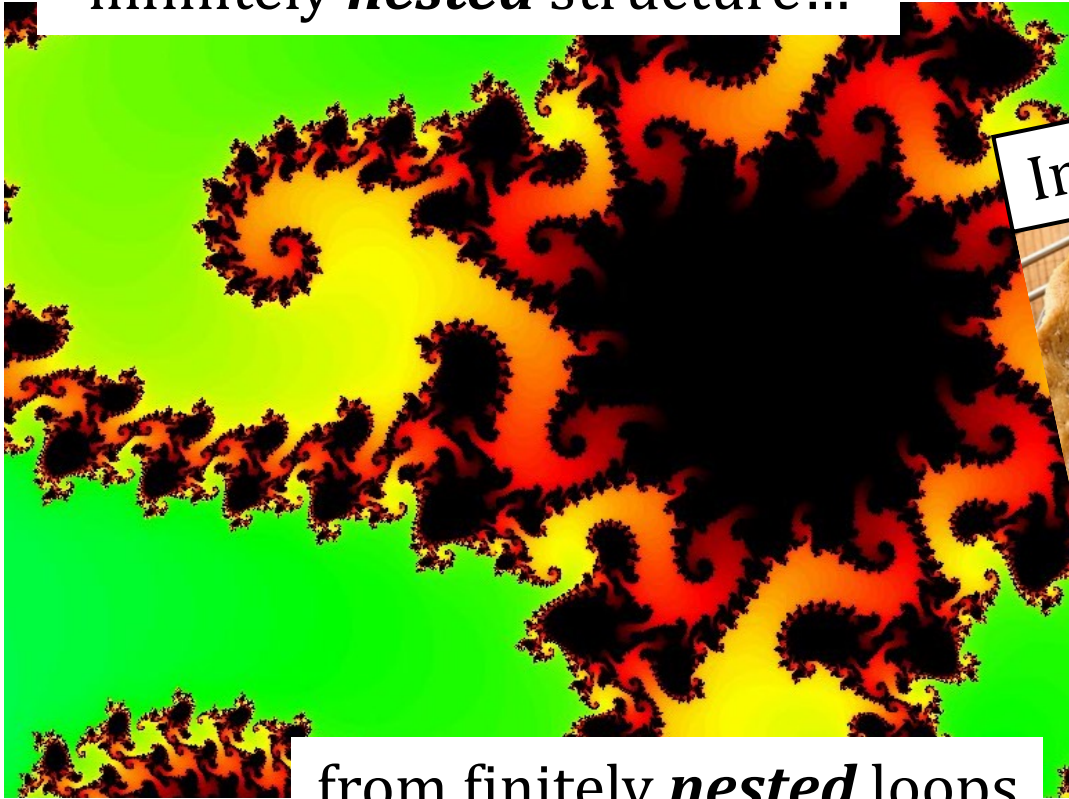




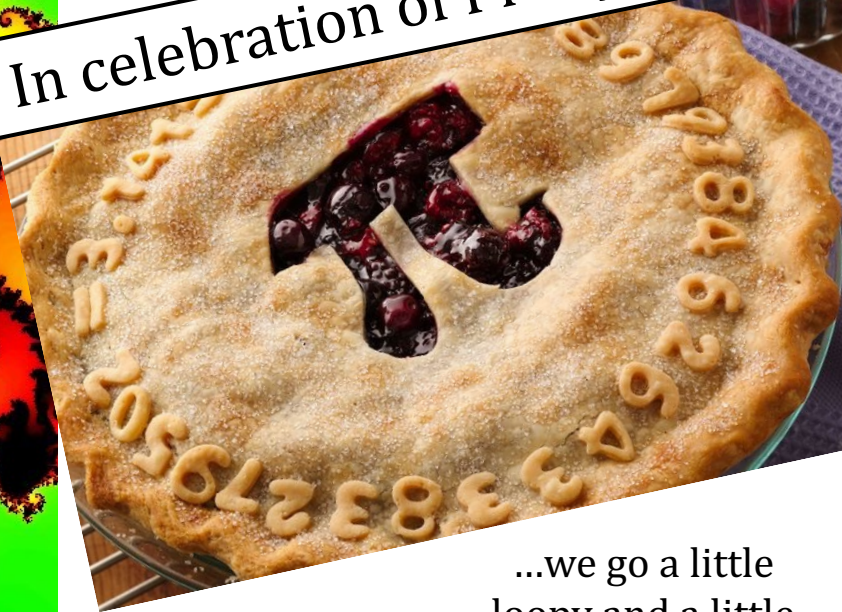
CS 5 Today

infinitely *nested* structure...



from finitely *nested* loops

In celebration of Pi day (3/14)



...we go a little
loopy and a little
random

Homework 6/7 ***Last part*** due **Today!**

Homework 8 ***Loops!*** due March 26

Today: *Thinking in loops*

for

while

Today: *Thinking in loops*

for

```
for x in listrange(42):  
    print(x)
```

while

```
x = 1  
while x < 42:  
    print(x)  
    x *= 2
```

*What are the **design** differences between these two types of Python loops?*

Loop design...

HOW TO BREATHE EASIER

- 1 Inhale deeply through your nose.
- 2 Hold your breath for 5 long seconds counting "**1-Mississippi, 2-Mississippi, etc.**"
- 3 **DO NOT EXHALE.** Breathe in another short breath and hold for 5 long seconds.
- 4 **DO NOT EXHALE.** Repeat one more time.
- 5 Exhale **SLOWLY** to a slow count of 10 long seconds.
- 6 Repeat the whole sequence until you feel the stress, anger or frustration exit your body.

Careful here!

Is this a
for or a
while
loop?

MORE TIPS?

www.cuc.claremont.edu/heo/balance

find your balance!




Table tent ... from a past year at the Hoch

Loop design...


HOW TO BREATHE EASIER

- 1 Inhale deeply through your nose.
- 2 Hold your breath for 5 long seconds counting "**1-Mississippi, 2-Mississippi, etc.**"
- 3 **DO NOT EXHALE.** Breathe in another short breath and hold for 5 long seconds.
- 4 **DO NOT EXHALE.** Repeat one more time. ←
- 5 Exhale **SLOWLY** to a slow count of 10 long seconds.
- 6 Repeat the whole sequence until you feel the stress, anger or frustration exit your body.

MORE TIPS?

www.cuc.claremont.edu/heo/balance

find your balance!



Are these
for or
while
loops?



Thinking in *loops*

for

*definite
iteration*

a.k.a “bounded iteration”

For a **known** *sequence*
(specific number of iterations)

while

*indefinite
iteration*

For an **unknown**
number of iterations

Homework 8 preview

#0

When Algorithms Discriminate...

#1 ~ lab

The Mandelbrot Set

#2

Lots of loops!

#3

Pi from Pie

#4

TTS Securities

(Extra)

ASCII Art

(Web extra)

CSS: Cascading Style Sheets

*Loopy
thinking*

A diagram illustrating the concept of 'Loopy thinking' as a central theme. A large orange arrow points from the right towards the center. From the tip of this arrow, five blue arrows point outwards to the following topics: 'The Mandelbrot Set', 'Lots of loops!', 'Pi from Pie', 'TTS Securities', and 'ASCII Art'. The text 'Loopy thinking' is written in blue italics to the right of the arrows.

π day!

3/14/15 9:26:53

The screenshot shows a Windows 7 desktop environment. The taskbar at the top contains several open browser tabs: 'Inbox (47) - zdodds@gmail.com', 'How to Survive the College', and 'Pi Day - Google Search'. The active browser window displays a Google search for 'Pi Day', showing approximately 128,000,000 results. The search results page highlights 'Saturday, March 14, Pi Day 2015' and includes a news snippet from USA TODAY about kids reciting 3.14. Overlaid on the browser window is a command prompt window titled 'C:\Windows\system32\cmd.exe - python'. It shows the execution of 'python' which displays the Python version (2.7.6) and the value of 'math.pi' (3.141592653589793). A second command prompt window is also open, showing an error message: 'ImportError: 'import' is not recognized as an internal or operable program or batch file.' In the bottom right corner, a system clock widget displays the date 'Saturday, March 14, 2015', a calendar for March, and the time '9:26:53 AM'. A large green banner with yellow text is positioned in the bottom left corner.

Google
Pi Day

Web Images Maps News Videos More Search tools

About 128,000,000 results (0.35 seconds)

Saturday, March 14
Pi Day 2015

In the news

Happy Pi Day! Watch these stunning videos of kids reciting 3.14
USA TODAY - 1 hour ago
This March 14 is a very special Pi Day that comes but a century. That's right, it's ...

Pi Day Isn't Just A...

```
C:\Windows\system32\cmd.exe - python
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Owner>import math
'import' is not recognized as an internal or operable program or batch file.

C:\Users\Owner>python
Python 2.7.6 (default, Nov 10 2013, 19:24:18)
Type "help", "copyright", "credits" or "license()"
>>> import math
>>> math.pi
3.141592653589793
>>>
```

March, 2015

Su	Mo	Tu	We	Th	Fr	Sa
22	23	24	25	26	27	28
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4

9:26:53 AM

Daylight Saving Time began on Sunday, March 08, 2015 at 2:00 AM. The clock went forward 1 hour at that time.

Change date and time settings...

9:26 AM
3/14/2015

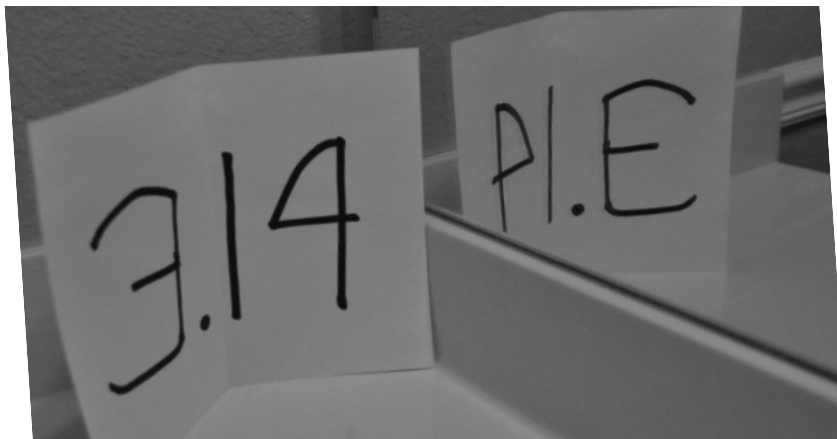
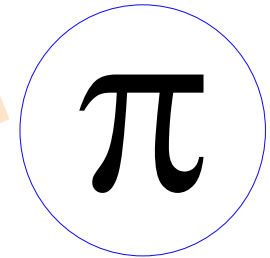
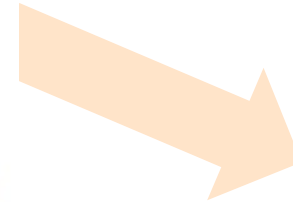
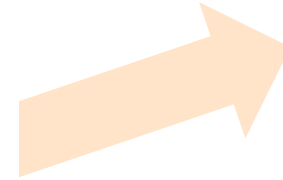
"Best π day" ever was in 2015!

Hw8 Pr3

Honoring "π day"

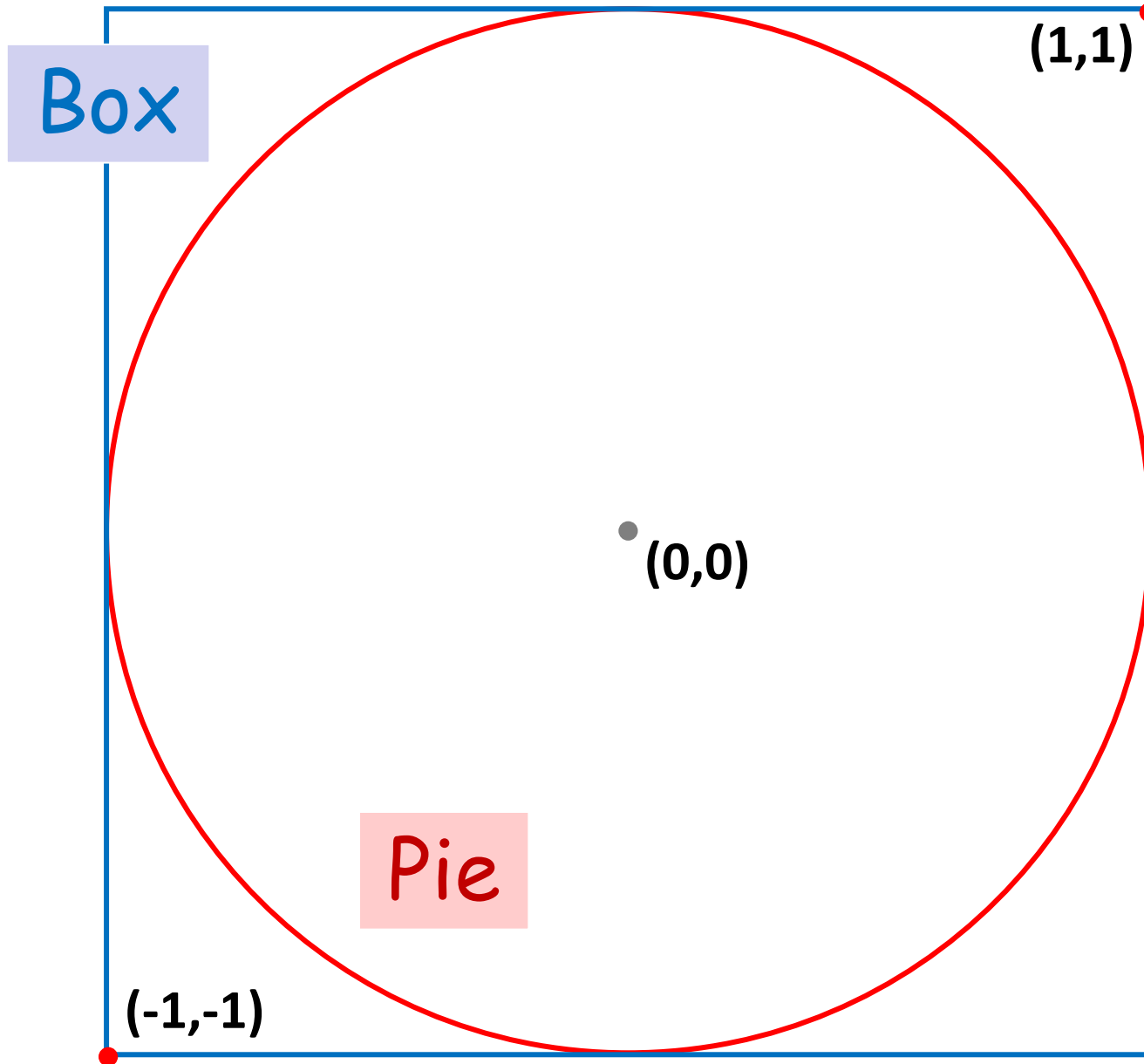
π from Pie?

Box



This *couldn't* be just a coincidence!





Estimating
 π from pie?

What if
we just
throw
darts at
this
picture?

Pi-design challenge...

Name(s) _____

(1,1)

Estimating π from pie?

(1) Suppose you **throw** 1000 darts at the **square**.
(All of them do hit the square.)

(2) Suppose 800 of them end up as **hits** in the circle.

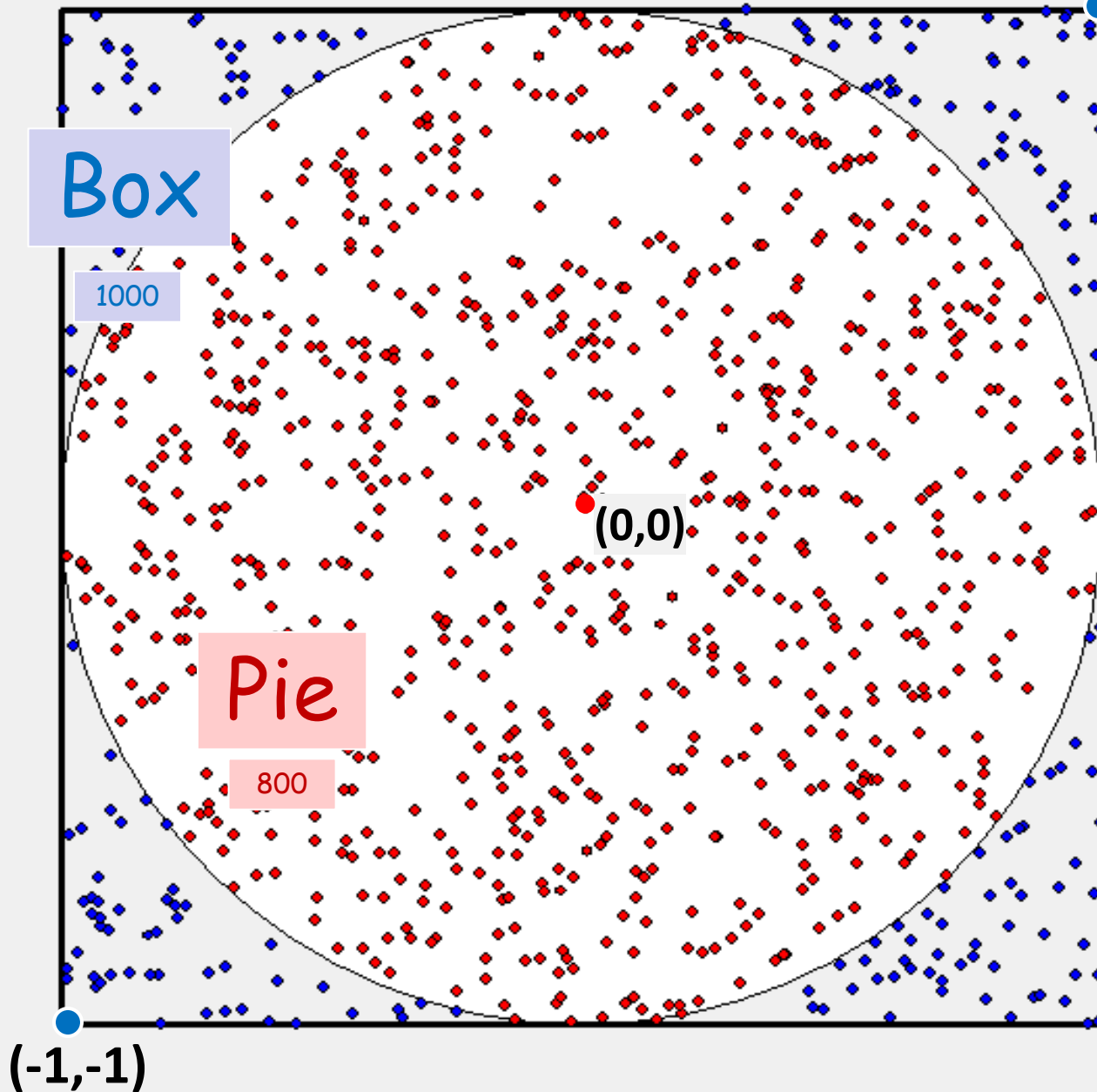
(3) What is the estimated value of π from this # of hits (800) & throws (1000)?

Hints

How big is a **side** of the square? its **area**?

How big is the **radius** of the circle? its **area**?

How do these help!?!?



Pi-design challenge...

Name(s) _____

(1,1)

Estimating
 π from pie?

Box

1000

Go full circle by
handing these back!

(-1,-1)

(1) Suppose
w
re.
re.)
n
cle.
I
(1000) & throws (1000)?

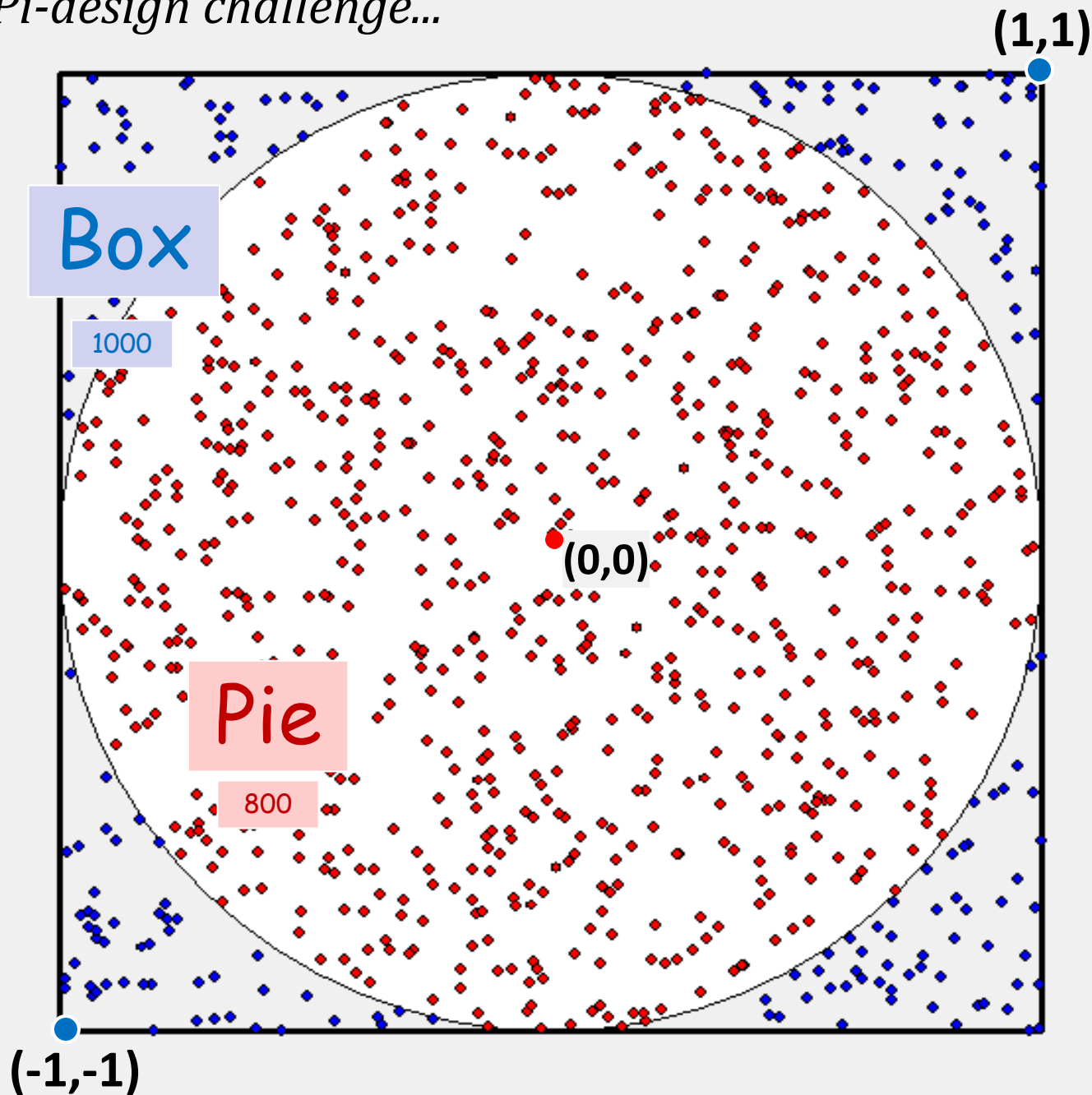
Hints

How big is a **side** of the square? its **area**?

How big is the **radius** of the circle? its **area**?

How do these help!?!?

Pi-design challenge...



Estimating π from pie?

$$\frac{\pi}{4} = \frac{\text{area}}{\text{area}}$$

$$\pi \sim \frac{4 * \text{hits}}{\text{hits}}$$

Hw8 Pr3

Loops: **for** or **while**?

`pi_one(e)`

*e == how close to π
we need to get*

`pi_two(n)`

*n == number of
darts to throw*

Which function will use which kind of loop?

Loops: **for** or **while**?

`pi_one(e)`

while

*e == how close to π
we need to get*

`pi_two(n)`

for

*n == number of
darts to throw*

There's a loop for all seasons!

Better π with less work?

Leibniz's formula...

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \dots = \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1},$$

Better π with less work?

Leibniz's formula...

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \dots = \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1},$$

```
def leibniz_pi(k):  
    total = 0  
    for i in range(k):  
        total += (-1)**i / (2*i+1)  
    return total * 4  
  
def leibnizpi_lc(k):  
    LC = [(-1)**i / (2*i+1) for i in range(k)]  
    return sum(LC) * 4
```

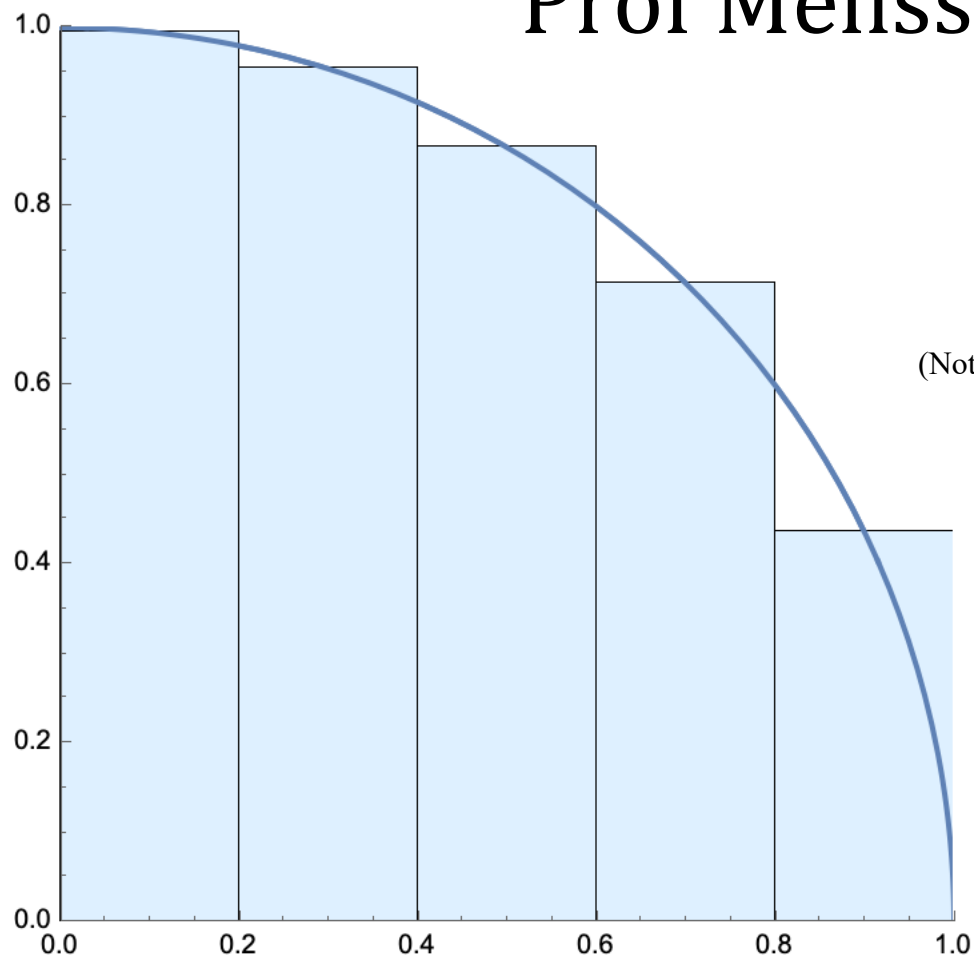


Better π with less work?

Prof Melissa's approach...

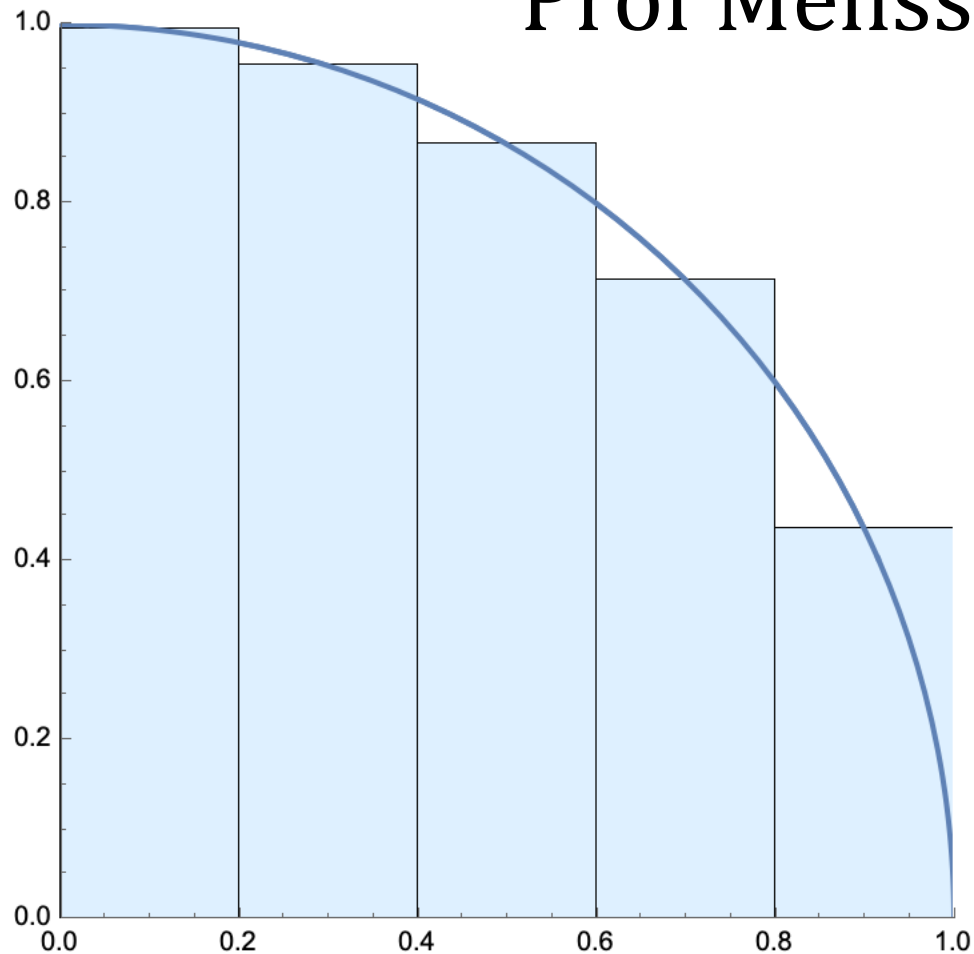
Figure out an easy way,
math genius not required!

(Not first to think of this way! Uses well-known math ideas!)



Better π with less work?

Prof Melissa's approach...



$$\sum_{i=0}^{k-1} \left(\frac{1}{k} \sqrt{1 - \left(\frac{i + \frac{1}{2}}{k} \right)^2} \right)$$

approximates this integral

$$\int_0^1 \sqrt{1 - x^2} dx$$

Better Pi with less work?

Prof Melissa's approach...

```
def slice_pi(k):  
    total = 0  
    width = 1/k  
    for i in range(k):  
        x = (i+0.5)/k  
        height = math.sqrt(1 - x**2)  
        total += height * width  
    return total * 4
```

$$\sum_{i=0}^{k-1} \left(\frac{1}{k} \sqrt{1 - \left(\frac{i + \frac{1}{2}}{k} \right)^2} \right)$$

```
def slice_pi_lc(k):  
    LC = [math.sqrt(1 - (i+0.5)/k**2)/k for i in range(k)]  
    return sum(LC) * 4
```


Better Pi with less work?

... how about a bit of Gauss!

The Gauss–Legendre iterative algorithm:

Initialize

$$a_0 = 1, \quad b_0 = \frac{1}{\sqrt{2}}, \quad t_0 = \frac{1}{4}, \quad p_0 = 1.$$

Iterate

$$a_{n+1} = \frac{a_n + b_n}{2}, \quad b_{n+1} = \sqrt{a_n b_n},$$

$$t_{n+1} = t_n - p_n (a_n - a_{n+1})^2, \quad p_{n+1} = 2p_n.$$

Then an estimate for π is given by

$$\pi \approx \frac{(a_n + b_n)^2}{4t_n}.$$

Better Pi with less work?

... how about a bit of Gauss!

The Gauss–Legendre iterative algorithm:

Initialize

$$a_0 = 1, \quad b_0 = \frac{1}{\sqrt{2}}, \quad t_0 = \frac{1}{4}, \quad p_0 = 1.$$

Iterate

$$a_{n+1} = \frac{a_n + b_n}{2}, \quad b_{n+1} = \sqrt{a_n b_n},$$

$$t_{n+1} = t_n - p_n(a_n - a_{n+1})^2, \quad p_{n+1} = 2p_n.$$

Then an estimate for π is given by

$$\pi \approx \frac{(a_n + b_n)^2}{4t_n}.$$

```
def gausspi(k):  
    a = 1  
    b = 1/(2**0.5)  
    t = 0.25  
    p = 1  
    for i in range(k):  
        a_next = (a + b) / 2  
        b = (a * b)**0.5  
        t -= p * (a - a_next)**2  
        a = a_next  
        p *= 2  
    return (a + b)**2 / (4 * t)
```

Homework 8 preview

#0

When Algorithms Discriminate...

#1 ~ lab

The Mandelbrot Set

#2

Lots of loops!

#3

Pi from Pie

#4

Text menus...

(EC5)

ASCII Art

*Thinking
in Loops...*



PythonBat loop practice...

google for "*PythonBat*" then...

CodingBatcode practice

[about](#) | [help](#) | [code help+videos](#) | [done](#) | [prefs](#)

[forgot password](#) | [create account](#)

Java

Python

String-2 > double_char

[prev](#) | [next](#) | [chance](#)

Given a string, return a string where for every char in the original, there are two chars.

double_char('The') → 'TThhee'
double_char('AAbb') → 'AAAAbbbb'
double_char('Hi-There') → 'HHii--TThheerree'

Go


...Save, Compile, Run (ctrl-enter)

Show Hint

```
def double_char(str):  
    result = ''  
    for c in str:  
        result += 2*c  
    return result
```

Our Solution:

```
def double_char(str):  
    result = ''  
    for i in range(len(str)):  
        result += str[i] + str[i]  
    return result
```

 All Correct

Good job -- problem solved. You can see our solution as an alternative.

10 required, up to +15 EC points available...

Homework 8 preview

#0

When Algorithms Discriminate...

#1 ~ lab

The Mandelbrot Set

#2

Lots of loops!

#3

Pi from Pie


#4

Text menus... *Thurs.*

(EC5)

ASCII Art

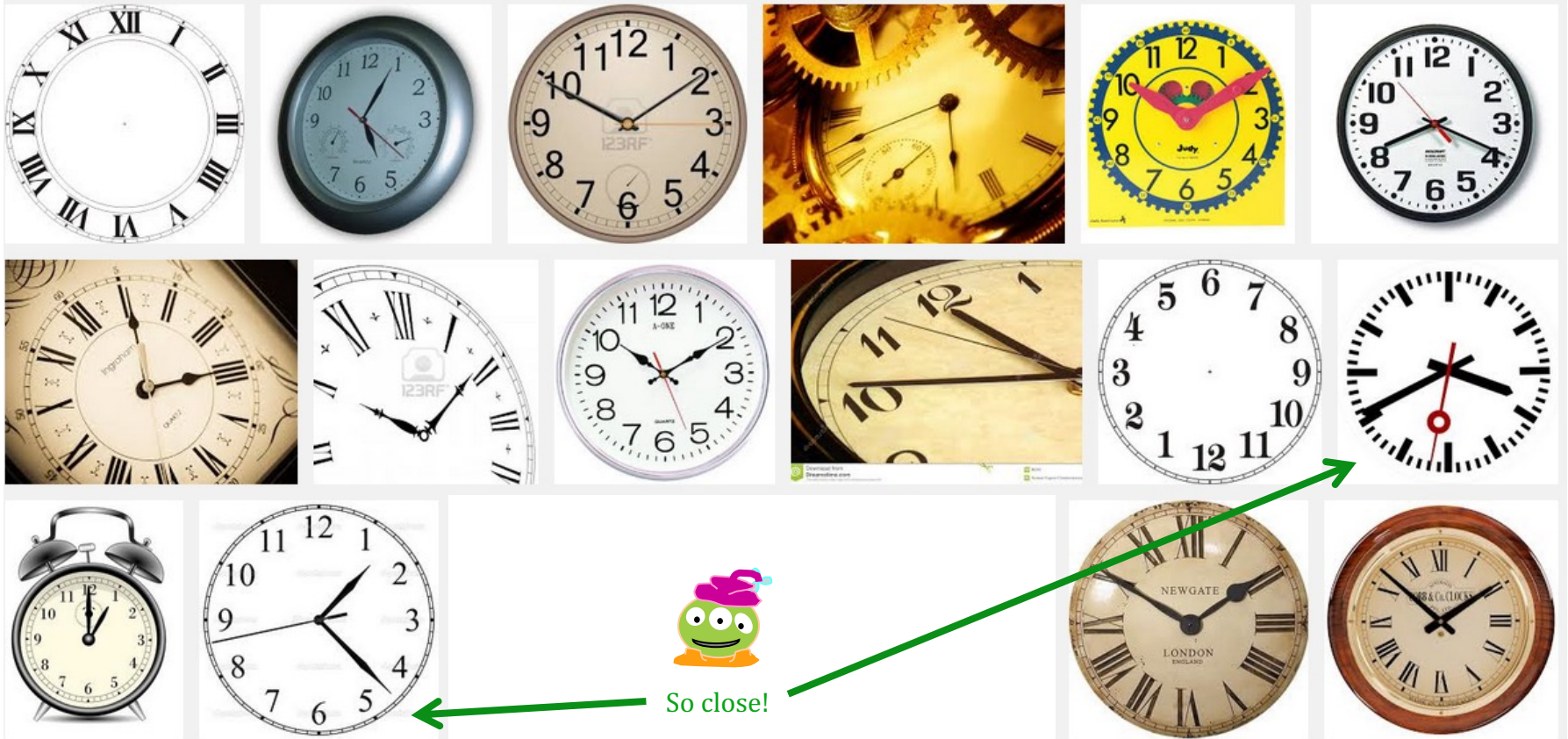
*Not just loops...
Nested loops*



Nested loops are familiar, too!

```
for mn in range(60):  
    for s in range(60):  
        tick()
```


Nested loops are familiar, too!



Nested loops

*Life
clock*



```
for y in range(years_in_life):  
    for m in range(12):  
        for d in range(f(m,y)):  
            for h in range(24):  
                for mn in range(60):  
                    for s in range(60):  
                        tick()
```

`mn == 0` → `s == 0` → `s == 59`

Double-Nested
loops'
2d structure

One hour ~ 3600 seconds

`mn == 42` →

`mn == 59` →

42nd minute's
59th second

```
for mn in range(60):  
    for s in range(60):  
        tick()
```

`hour()`

Creating 2D structure ~ in ASCII

		col			
		0	1	2	3
row	0	#	#	#	#
	1	#	#	#	#
	2	#	#	#	#

```
for row in range(3):  
    for col in range(4):  
        print("#")
```

*Wait! this needs
something more...*



Creating 2D structure ~ in ASCII

		col			
		0	1	2	3
row	0	#	#	#	#
	1	#	#	#	#
	2	#	#	#	#

```
for row in range(3):  
    for col in range(4):  
        print("#", end=' ')
```



Creating 2D structure

```
[0,1,2]  
for row in range(3):  
    [0,1,2,3]  
    for col in range(4):  
        print('#', end=' ')  
  
print()
```



row = 0
col = 0
col = 1
col = 2
col = 3

row = 1
col = 0
col = 1
col = 2
col = 3

row = 2
col = 0
col = 1
col = 2
col = 3

		col			
		0	1	2	3
row	0	#	#	#	#
	1	#	#	#	#
	2	#	#	#	#

Creating 2d structure

Let's take an
alien's-eye view!



```
for row in range(3):  
    for col in range(4):  
        if col == row:  
            print('#', end='')  
        else:  
            print(' ', end='')  
    print()
```

		col			
		0	1	2	3
row	0				
	1				
	2				

row = 0
col = 0
col = 1
col = 2
col = 3

row = 1
col = 0
col = 1
col = 2
col = 3

row = 2
col = 0
col = 1
col = 2
col = 3

Match!

What if-test will create the fourth, unmatched ASCII pattern?

if

* and ** are extra!

A

```
for r in range(3):
    for c in range(6):
        if c >= r:
            print('#', end='')
        else:
            print(' ', end='')
    print()
```

B

```
for r in range(3):
    for c in range(6):
        if c%2 == 1:
            print('#', end='')
        else:
            print(' ', end='')
    print()
```

C

```
for r in range(3):
    for c in range(6):
        if c%2 == r%2:
            print('#', end='')
        else:
            print(' ', end='')
    print()
```

1

	0	1	2	3	4	5
0	#	#	#	#	#	
1	#	#	#	#		
2	#	#	#			

rows

2

	0	1	2	3	4	5
0	#		#		#	
1		#		#		#
2	#		#		#	

rows

	0	1	2	3	4
0	#	#	#	#	#
1		#	#	#	
2			#		

rows

3

	0	1	2	3	4	5
0	#	#	#	#	#	#
1		#	#	#	#	#
2			#	#	#	#

rows

4

	0	1	2	3	4	5
0		#		#		#
1		#		#		#
2		#		#		#

rows

	0	1	2	3	4
0		#	#	#	
1	#		#		#
2	#	#		#	#

rows

if

if

Match!

What code creates the fourth, unmatched ASCII pattern?

if r+c < 5 :

D

answers...

A

```
for r in range(3):[0,1,2]
    for c in range(6):
        if c >= r:[0,1,2,3,4,5]
            print('#', end='')
        else:
            print(' ', end='')
    print()
```

B

```
for r in range(3):
    for c in range(6):
        if c%2 == 1:
            print('#', end='')
        else:
            print(' ', end='')
    print()
```

C

```
for r in range(3):
    for c in range(6):
        if c%2 == r%2:
            print('#', end='')
        else:
            print(' ', end='')
    print()
```

1

	0	1	2	3	4	5
0	#	#	#	#	#	
1	#	#	#	#		
2	#	#	#			

rows

if c+r<=4

2

C

	0	2	3	4	5
0	#		#		#
1		#		#	#
2	#		#		#

rows

	0	1	2	3	4
0	#	#	#	#	#
1		#	#	#	
2			#		

if c+r<=4 **and** c>=r

3

A

	0	2	3	4	5
0	#	#	#	#	#
1		#	#	#	#
2			#	#	#

rows

4

B

	0	2	3	4	5
0		#		#	#
1		#		#	#
2		#		#	#

rows

	0	1	2	3	4
0		#	#	#	
1	#		#		#
2	#	#		#	#

if not (c==r **or** c+r==4)

loops: from ASCII Art



That's my **type**
of alien!

[illegible]

From Lucien (they/them) to Everyone: 8:46 AM

https://www.youtube.com/watch?v=DEqXNfs_HhY

https://www.youtube.com/watch?v=DEqXNfs_HhY

... to "*real*" images!

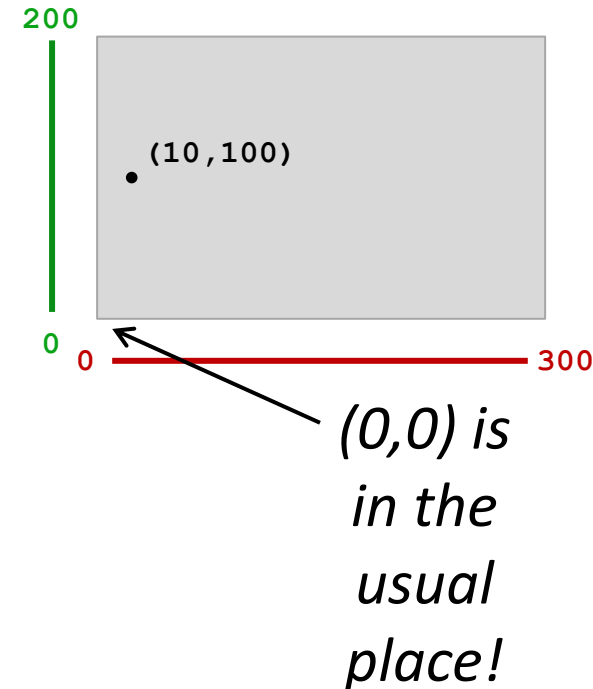
Python and images

```
from cs5png import *
```

*inputs are **width** and **height***

```
im = PNGImage( 300, 200 )
```

```
im.plotPixel( 10, 100 )
```

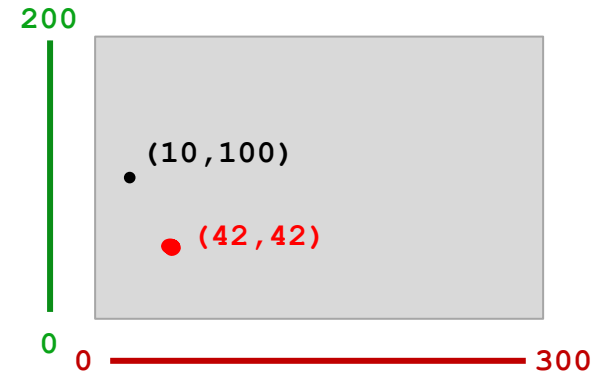


Python and images

```
from cs5png import *
```

*inputs are **width** and **height***

```
im = PNGImage( 300, 200 )
```



objects are new types that can contain their own functions, often called ***methods***

```
im.plotPoint( 10, 100 )
```

```
im.plotPoint( 42, 42, (255,0,0) )
```

col x row y red green blue

```
im.saveFile( )
```



These functions are clearly ***plotting something*** – if only I knew what they were up to...

```
from cs5png import *
```

```
def testImage():
```

```
    """ image demonstration """
```

```
    WD = 300
```

```
    HT = 200
```

```
    im = PNGImage( WD, HT )
```

```
    for row in range(HT):
```

```
        for col in range(WD):
```

```
            if col == row:
```

```
                im.plotPoint( col, row )
```

```
    im.saveFile()
```

Imagining Images

thicker line?
other diagonal?
stripes ?
thicker stripes?
thatching?

Complex Numbers!

Nothing's too
complex for
Python!



$$\sqrt{-1} = i$$

i can't believe this!

$$1j * 1j == -1$$

DOES ANY OF THIS REALLY HAVE TO DO WITH THE SQUARE ROOT OF -1? OR DO MATHEMATICIANS JUST THINK THEY'RE TOO COOL FOR REGULAR VECTORS?



COMPLEX NUMBERS AREN'T JUST VECTORS. THEY'RE A PROFOUND EXTENSION OF REAL NUMBERS, LAYING THE FOUNDATION FOR THE FUNDAMENTAL THEOREM OF ALGEBRA AND THE ENTIRE FIELD OF COMPLEX ANALYSIS.



AND WE'RE TOO COOL FOR REGULAR VECTORS.

I KNEW IT!

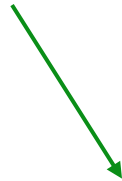


Complex Numbers!

$$\sqrt{-1} = i$$

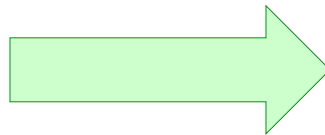


i *can't believe this!*



```
In[]: c = -2+1j
```

```
In[]: c**2
```



```
1j * 1j == -1
```

```
(-2+1j)*(-2+1j)
```

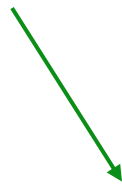
```
(3-4j)
```

Complex Numbers!

Nothing's too
complex for
Python!

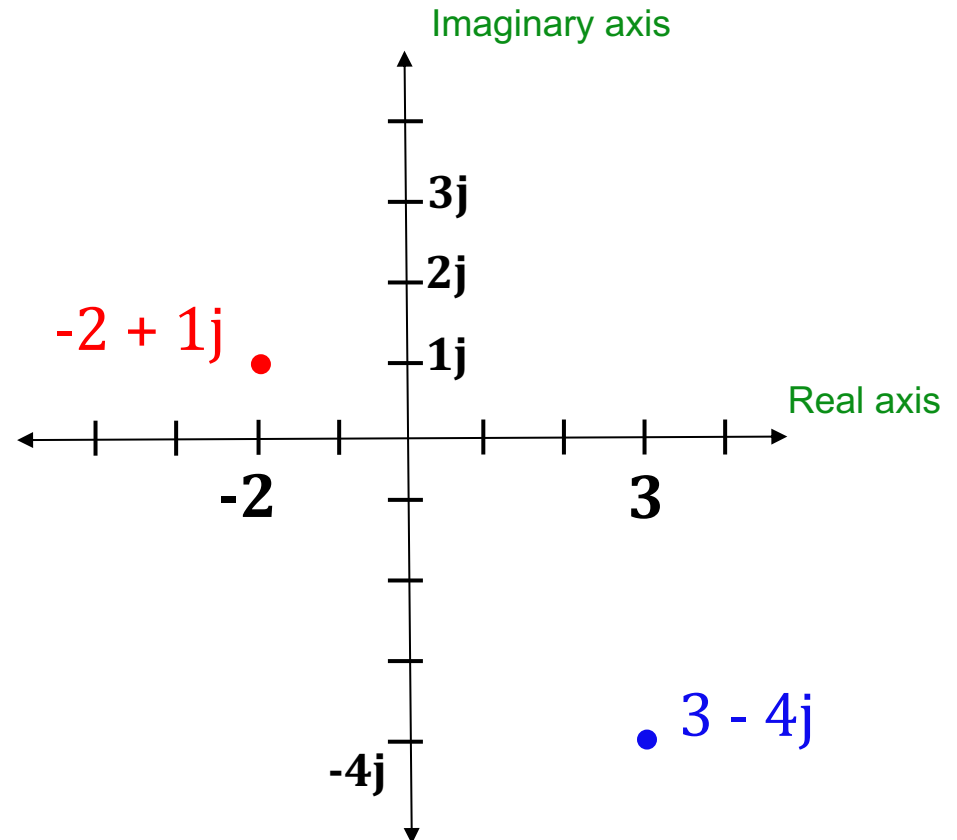


i can't believe this!



```
In[]: c = -2+1j
```

```
In[]: c**2  
(3-4j)
```

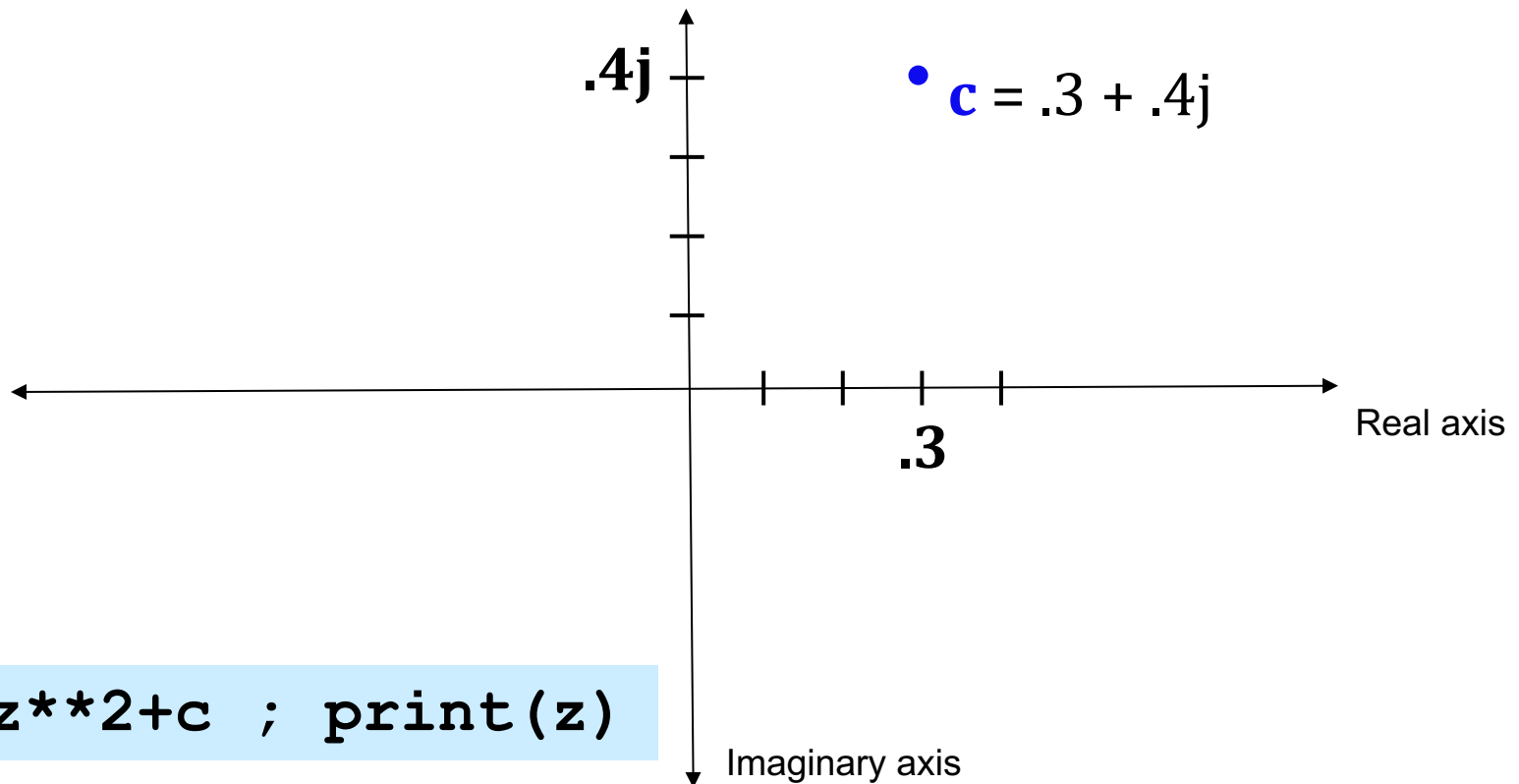


Lab 8: *the Mandelbrot Set*

Consider an *update rule*
for all complex numbers c

$$z_0 = 0$$

$$z_{n+1} = z_n^2 + c$$



```
z = z**2+c ; print(z)
```

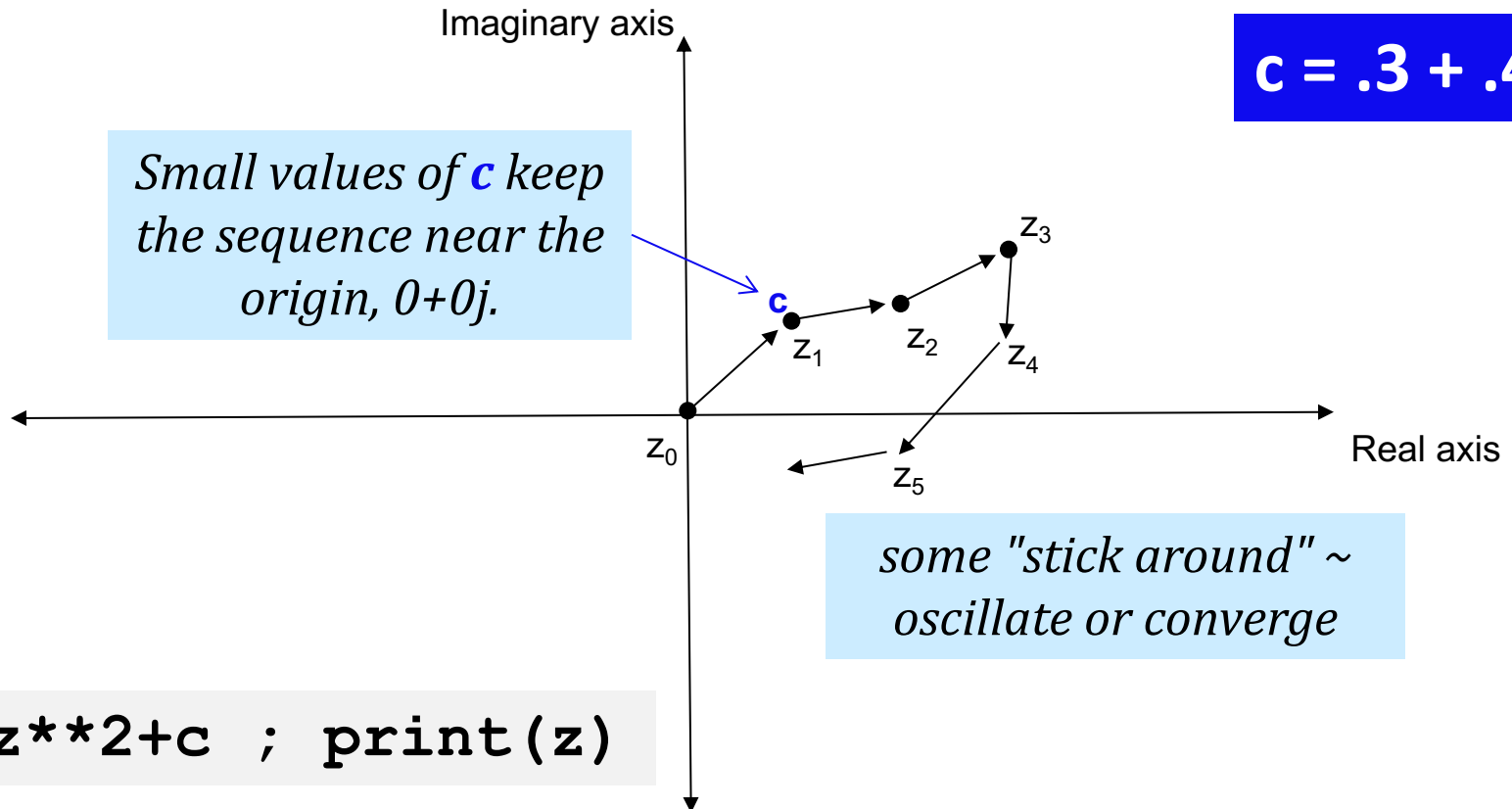
Mandelbrot Definition

Consider an *update rule*
for all complex numbers c

$$z_0 = 0$$

$$z_{n+1} = z_n^2 + c$$

$$c = .3 + .4j$$



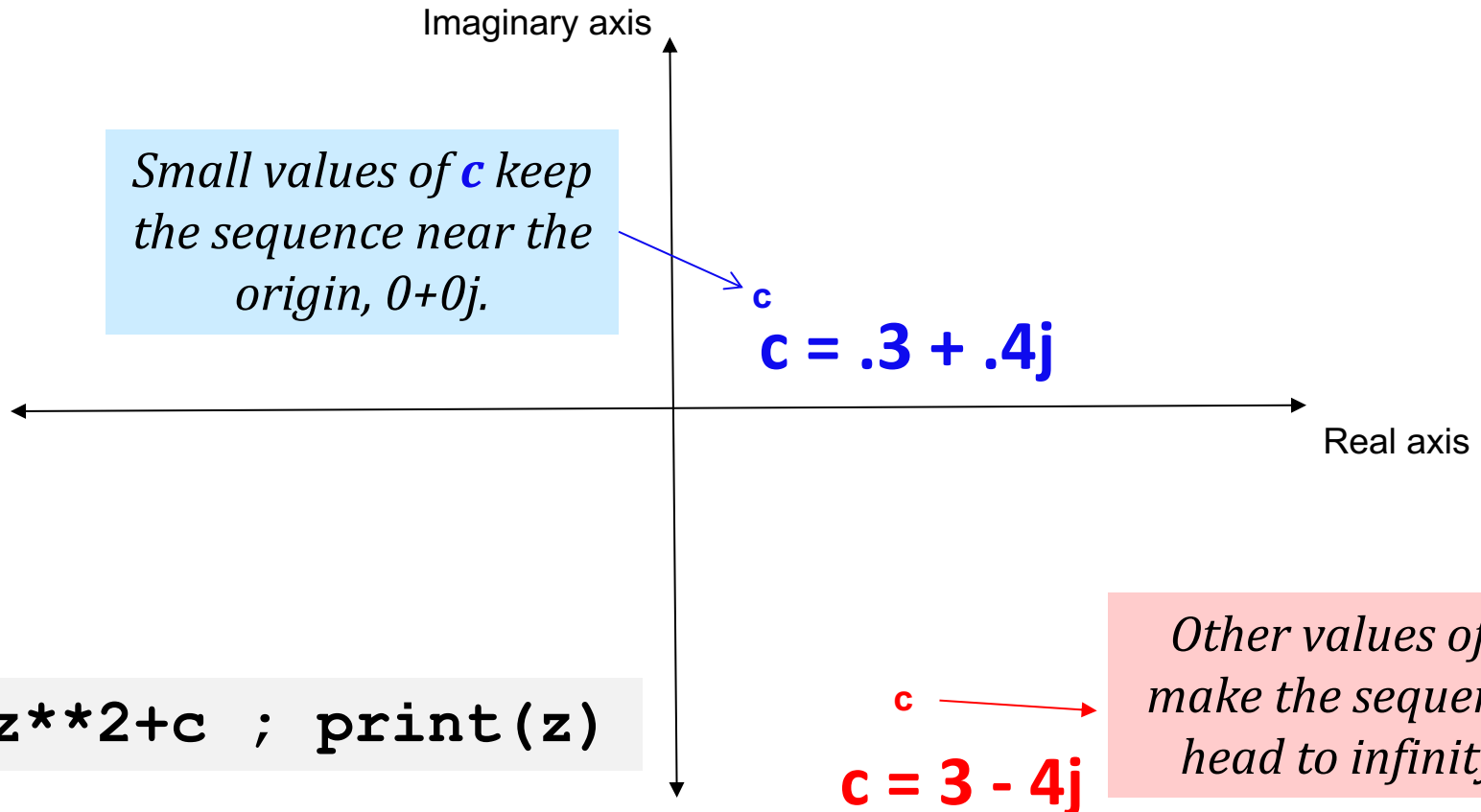
```
z = z**2+c ; print(z)
```

Mandelbrot Definition

Consider an *update rule*
for all complex numbers c

$$z_0 = 0$$

$$z_{n+1} = z_n^2 + c$$



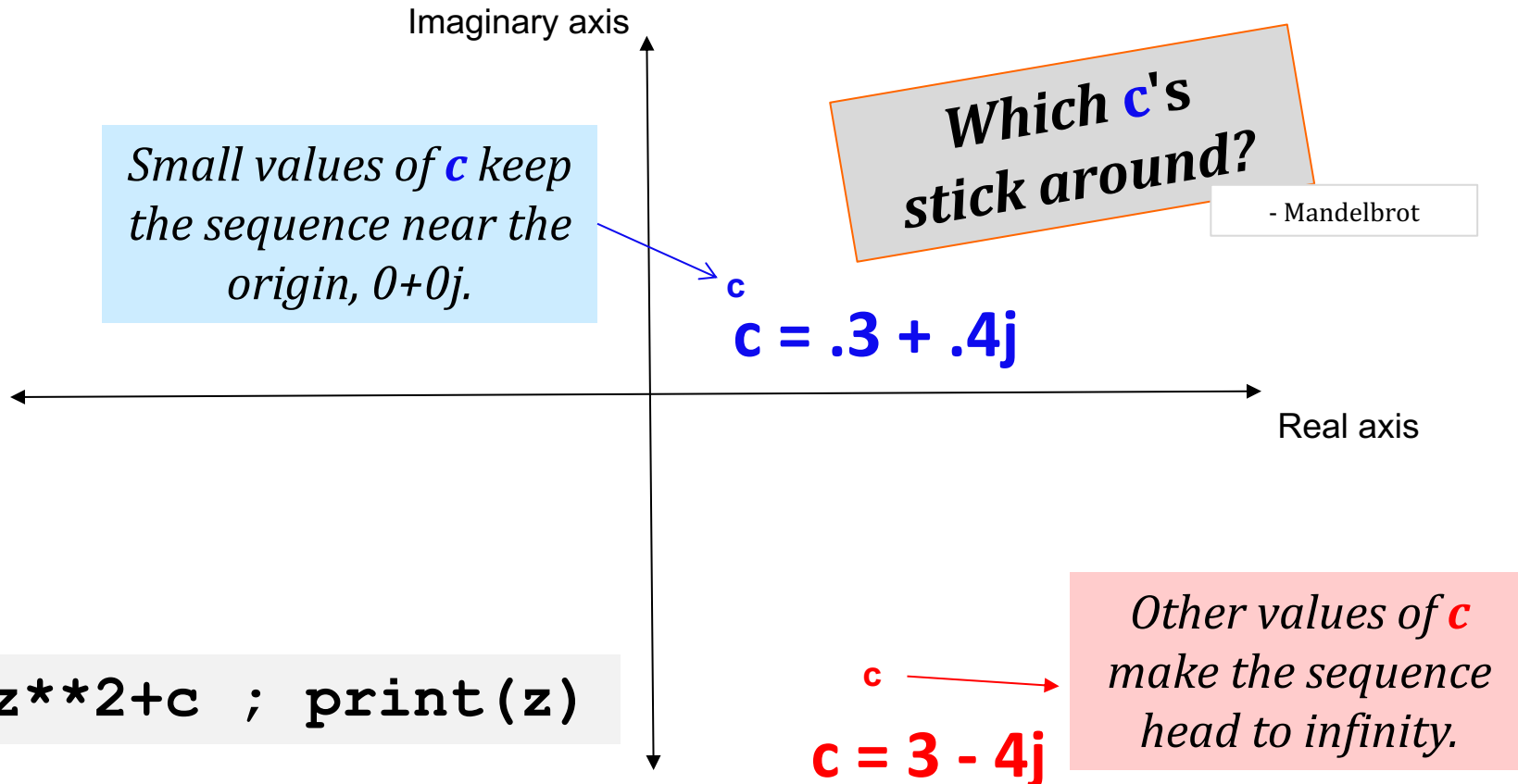
```
z = z**2+c ; print(z)
```

Mandelbrot Definition

Consider an *update rule*
for all complex numbers c

$$z_0 = 0$$

$$z_{n+1} = z_n^2 + c$$

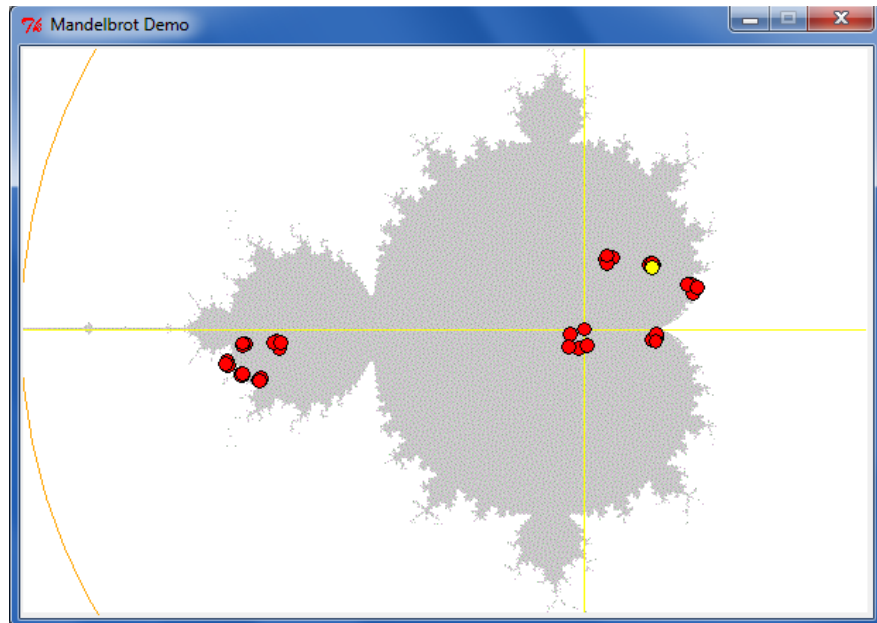


Lab 8: *the Mandelbrot Set*

Consider an *update rule*
for all complex numbers c

$$z_0 = 0$$

$$z_{n+1} = z_n^2 + c$$



Click to choose c .

c is $-1.21368948247 + -0.16290726817 * 1j$

```
iter # 0 : z = 0.0 + 0.0 * 1j
iter # 1 : z = -1.21368948247 + -0.16290726817 * 1j
iter # 2 : z = 0.232813899367 + 0.232530407823 * 1j
iter # 3 : z = -1.21355756129 + -0.0546346462374 * 1j
iter # 4 : z = 0.256047527535 + -0.0303026920702 * 1j
iter # 5 : z = -1.14904739926 + -0.17842516935 * 1j
iter # 6 : z = 0.0747849173552 + 0.07964 * 1j
iter # 7 : z = -1.26917000000 + 0.5977 * 1j
iter # 8 : z = -0.88 * 1j
iter # 9 : z = 0.08 * 1j
iter # 10 : z = 0.056431 * 1j
iter # 11 : z = 0.602 * 1j
iter # 12 : z = 0.14 * 1j
iter # 13 : z = 0.5207 * 1j
iter # 14 : z = 0.227677689 * 1j
iter # 15 : z = -0.180471770237 * 1j
iter # 16 : z = 0.255350697358 * 1j
iter # 17 : z = -1.26957426034 + -0.113606194429 * 1j
iter # 18 : z = 0.385222952638 + 0.125555732355 * 1j
iter # 19 : z = -1.08105700116 + -0.0661733682935 * 1j
iter # 20 : z = -0.0493841573866 + -0.0198329020025 * 1j
iter # 21 : z = -1.21164403147 + -0.160948405863 * 1j
iter # 22 : z = 0.228487387181 + 0.227117082506 * 1j
```

some c 's
stick around

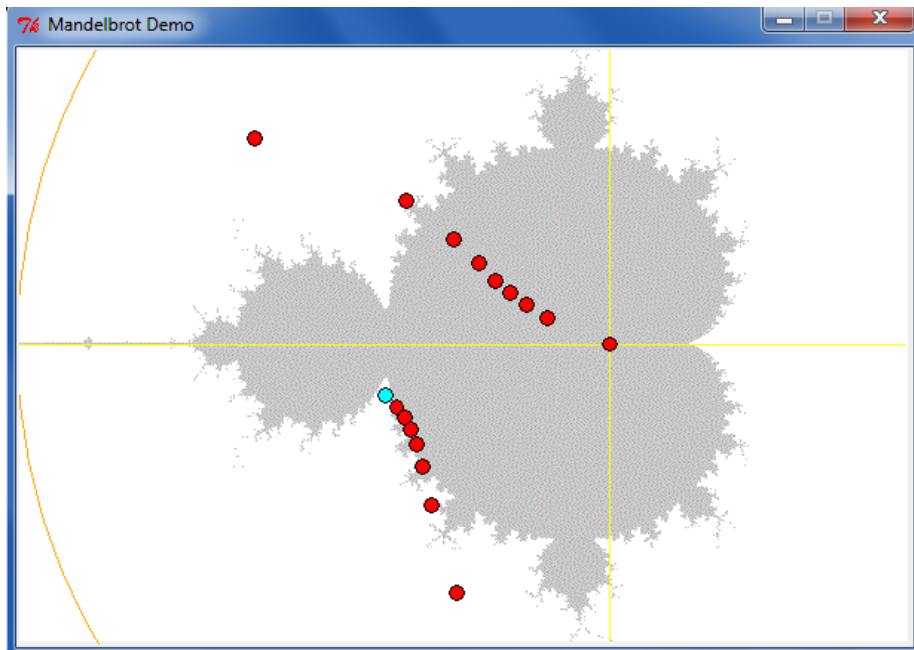


Lab 8: *the Mandelbrot Set*

Consider an *update rule*
for all complex numbers c

$$z_0 = 0$$

$$z_{n+1} = z_n^2 + c$$



Click to choose c .

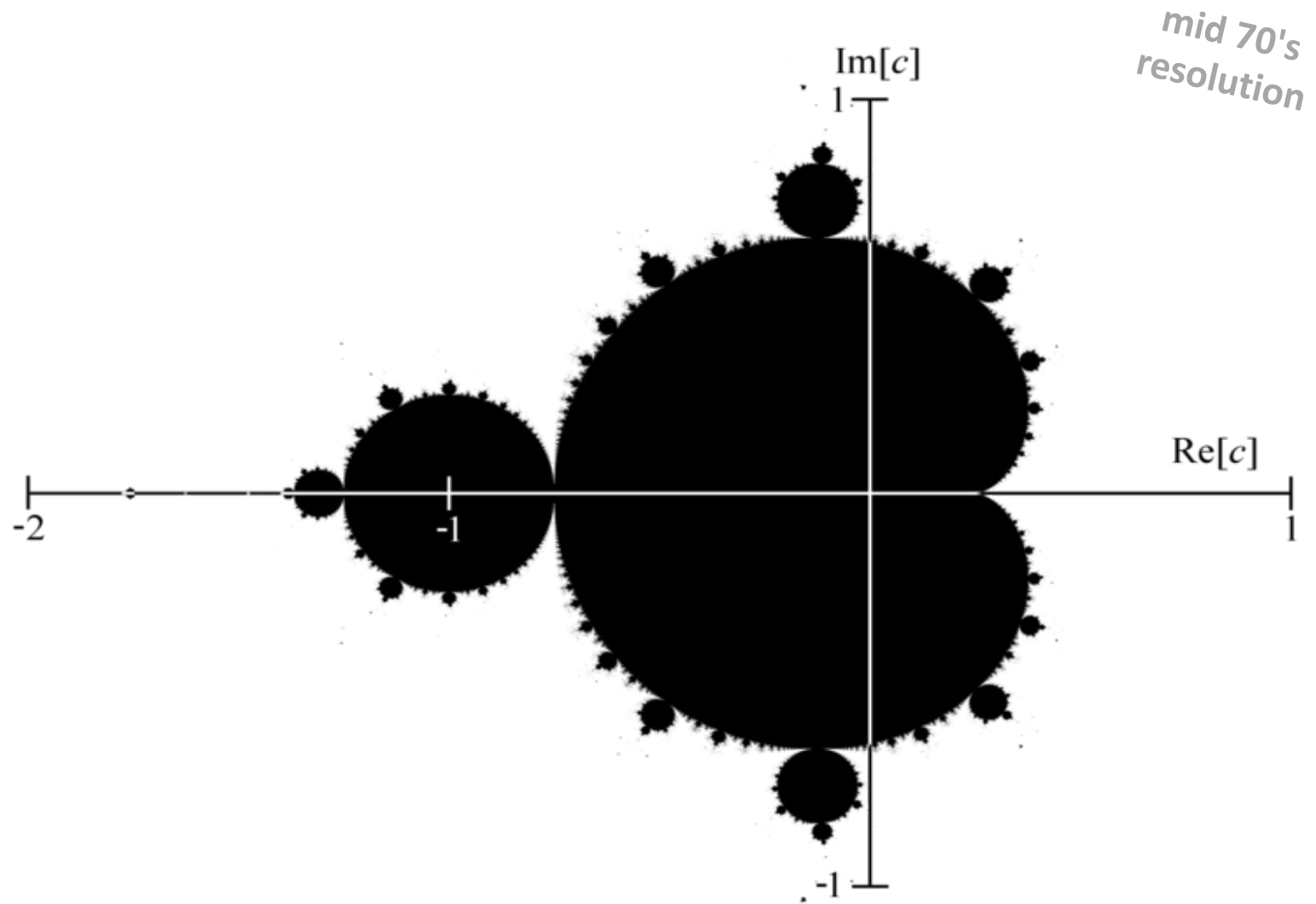
c is $-0.757929883139 + -0.172932330827 * 1j$

```
iter # 0 : z = 0.0 + 0.0 * 1j
iter # 1 : z = -0.757929883139 + -0.172932330827 * 1j
iter # 2 : z = -0.213377766429 + 0.0892088317622 * 1j
iter # 3 : z = -0.720358027597 + -0.211002693361 * 1j
iter # 4 : z = -0.283536331822 + 0.131000537188 * 1j
iter # 5 : z = -0.694714446542 + 0.369601 * 1j
iter # 6 : z = -0.336400000000 + 0.34238 * 1j
iter # 7 : z = -0.000000000000 + 0.197352 * 1j
iter # 8 : z = -0.000000000000 + 0.9922 * 1j
iter # 9 : z = -0.000000000000 + 0.50516 * 1j
iter # 10 : z = -0.000000000000 + 0.34007 * 1j
iter # 11 : z = -0.000000000000 + 0.0519 * 1j
iter # 12 : z = -0.000000000000 + 0.438 * 1j
iter # 13 : z = -0.000000000000 + 0.543792492805 * 1j
iter # 14 : z = -0.000000000000 + 0.48336747636 * 1j
iter # 15 : z = -0.000000000000 + -0.839488323663 * 1j
iter # 16 : z = -1.19623408871 + 0.693713130081 * 1j
iter # 17 : z = 0.191808204996 + -1.8326189188 * 1j
iter # 18 : z = -4.07963159717 + -0.875955021339 * 1j
iter # 19 : z = 15.1181668861 + 6.97421523468 * 1j
iter # 20 : z = 179.161361972 + 210.701767303 * 1j
```

other c 's
diverge



Mandelbrot Set ~ *points that stick around*



The shaded area are points that do **not** diverge for $z = z^{**2} + c$

Higher-resolution M. Set

$-2 + 1j$

connected

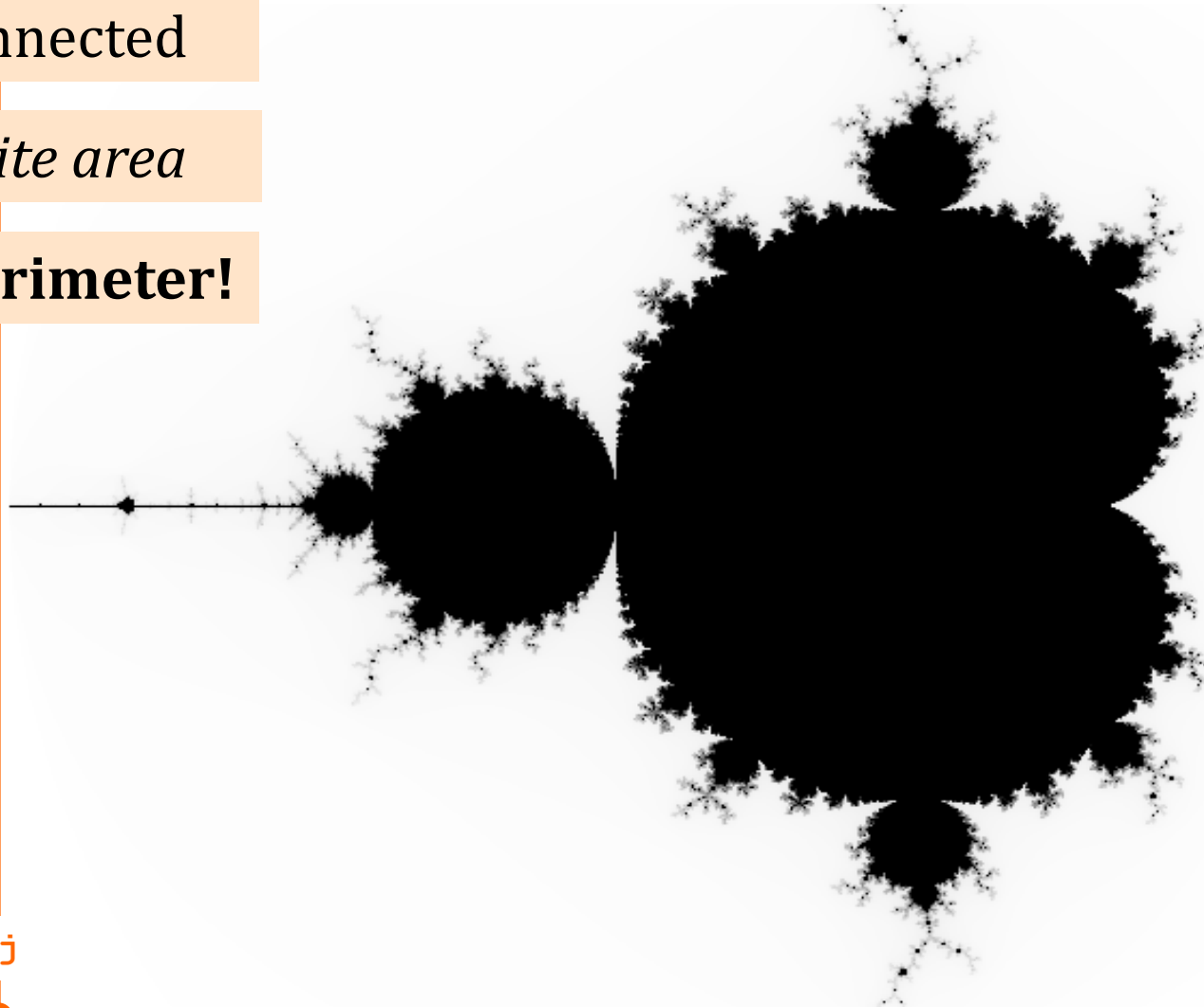
finite area

∞ **perimeter!**

$1 + 1j$

$-2 - 1j$

$1 - 1j$



The black pixels are points that do ***not*** diverge for $z = z^2 + c$

Chaos?

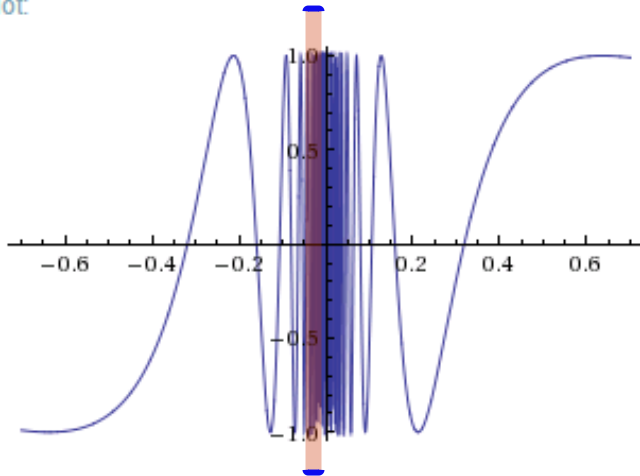
Input interpretation:

plot

$$y = \sin\left(\frac{1}{x}\right)$$

$x = -0.7$ to 0.7

Plot:



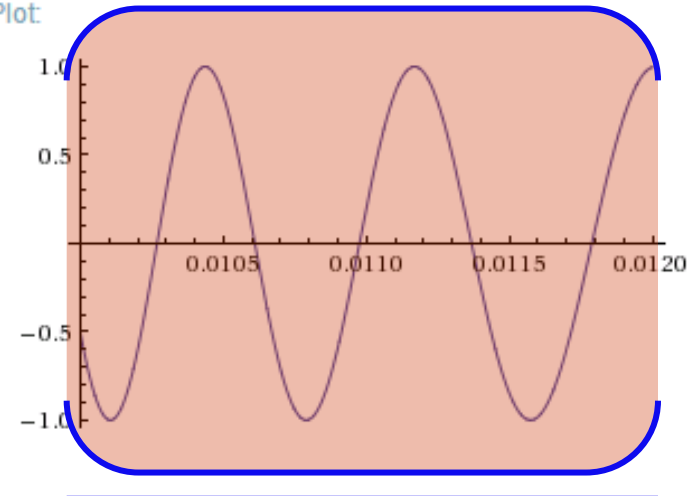
Input interpretation:

plot

$$y = \sin\left(\frac{1}{x}\right)$$

$x = 0.01$ to 0.012

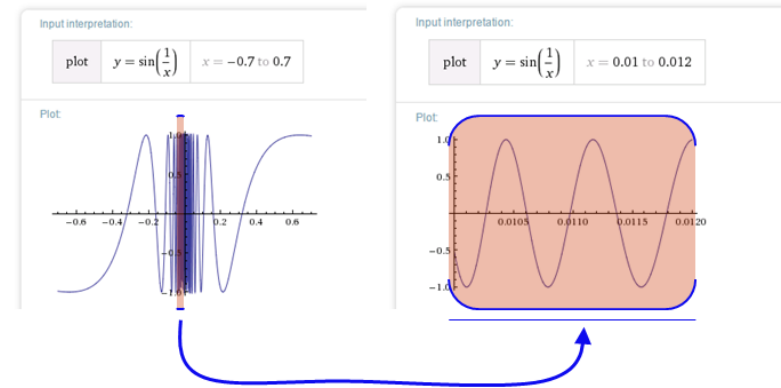
Plot:



Complex things always consisted of simple parts...

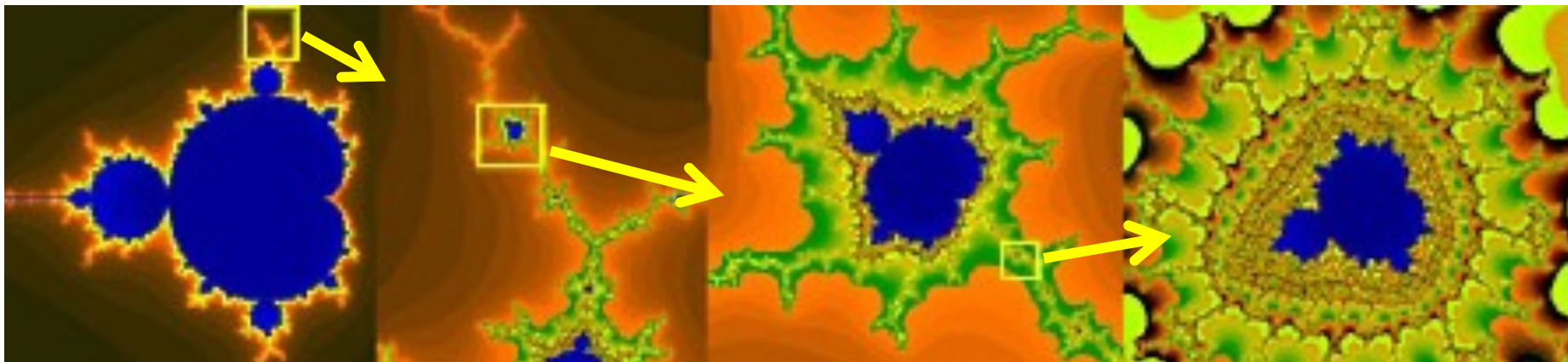
Before the M. Set, complex things
were made of simple parts:

Chaos!

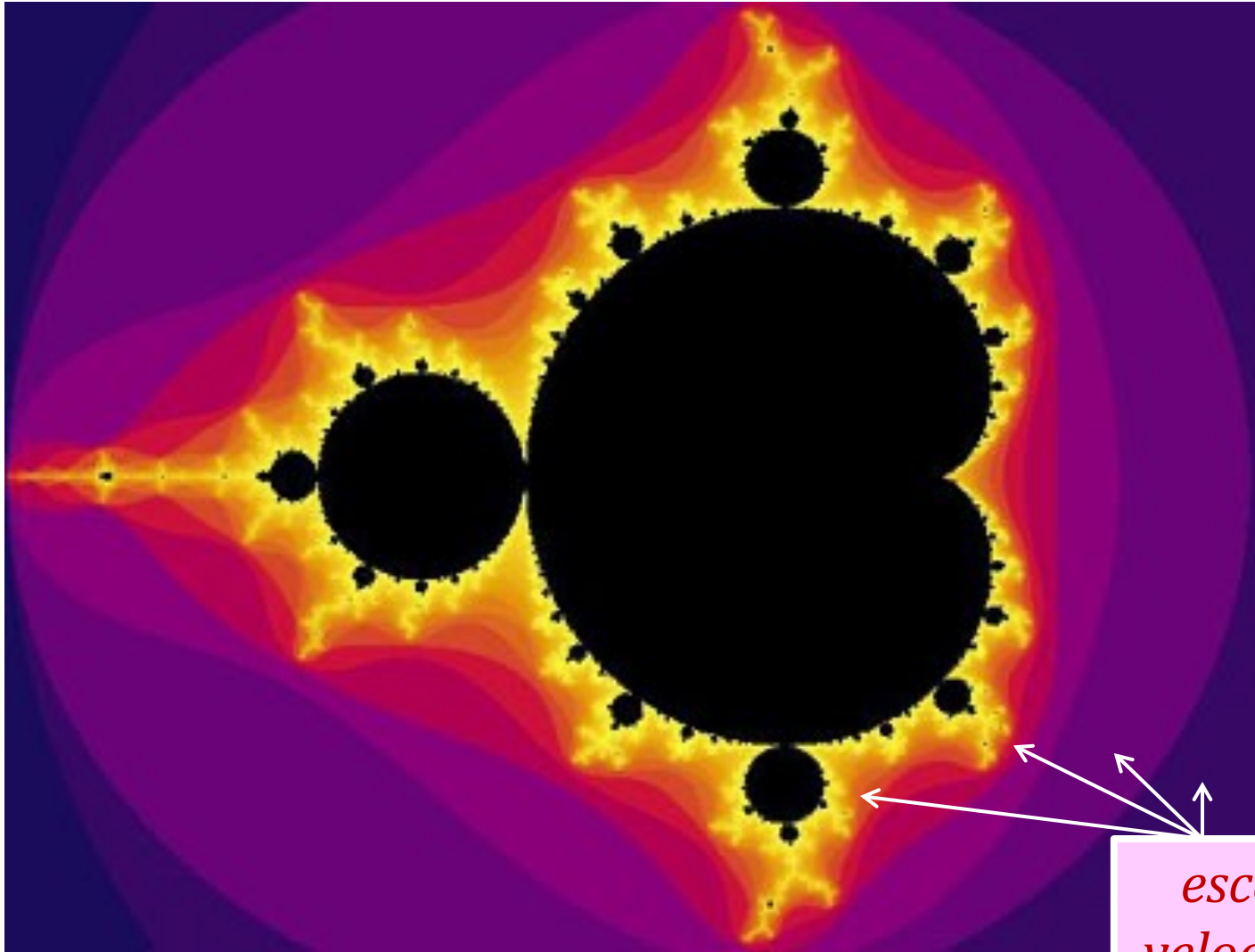


This was a "*naturally occurring*" object where
zooming uncovers **more** detail, not less:

not self-similar, but **quasi**-self-similar



The M. Set pixels are points that do not diverge for $z = z^2 + c$

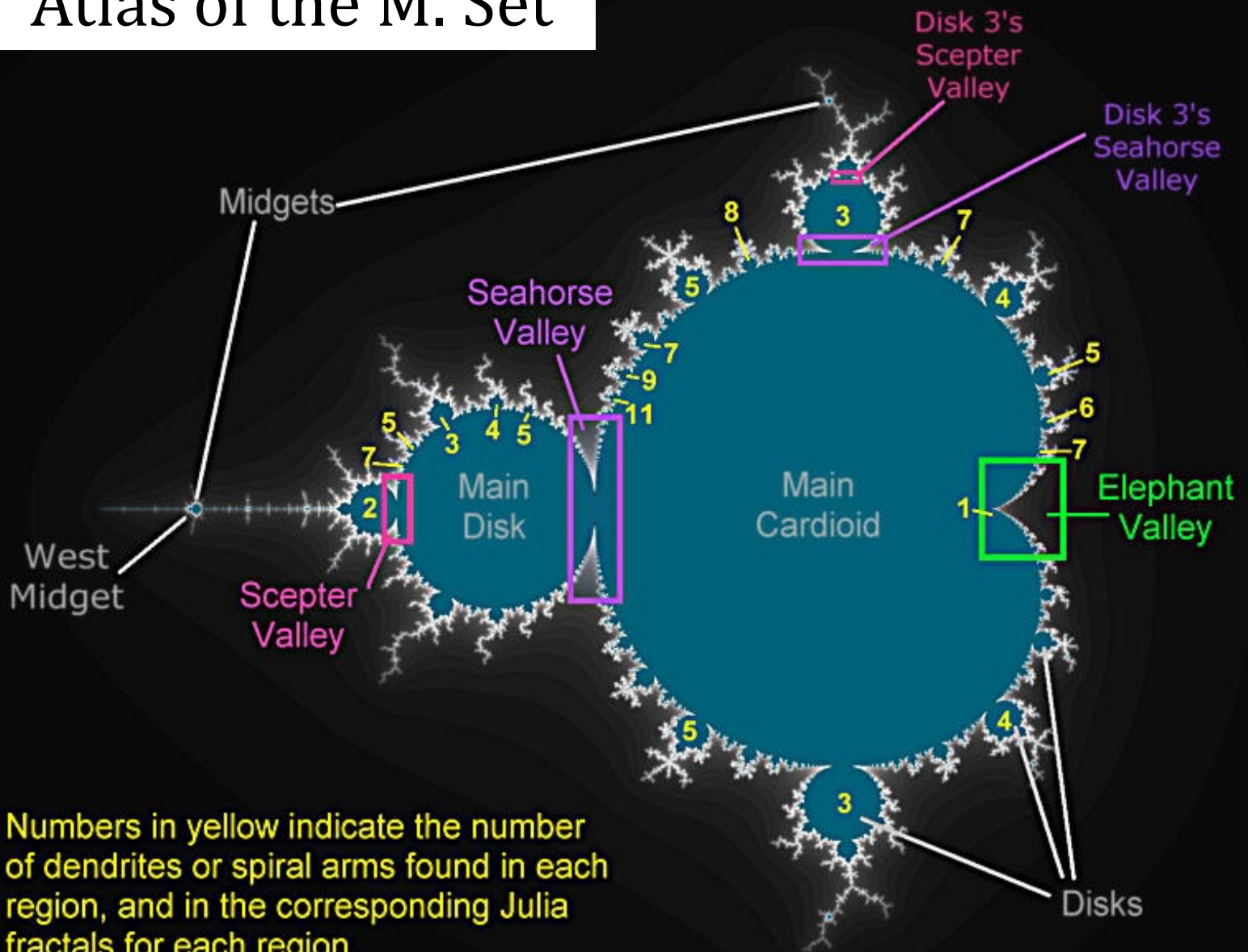


*escape
velocities!*

What are these colors?

??

Atlas of the M. Set

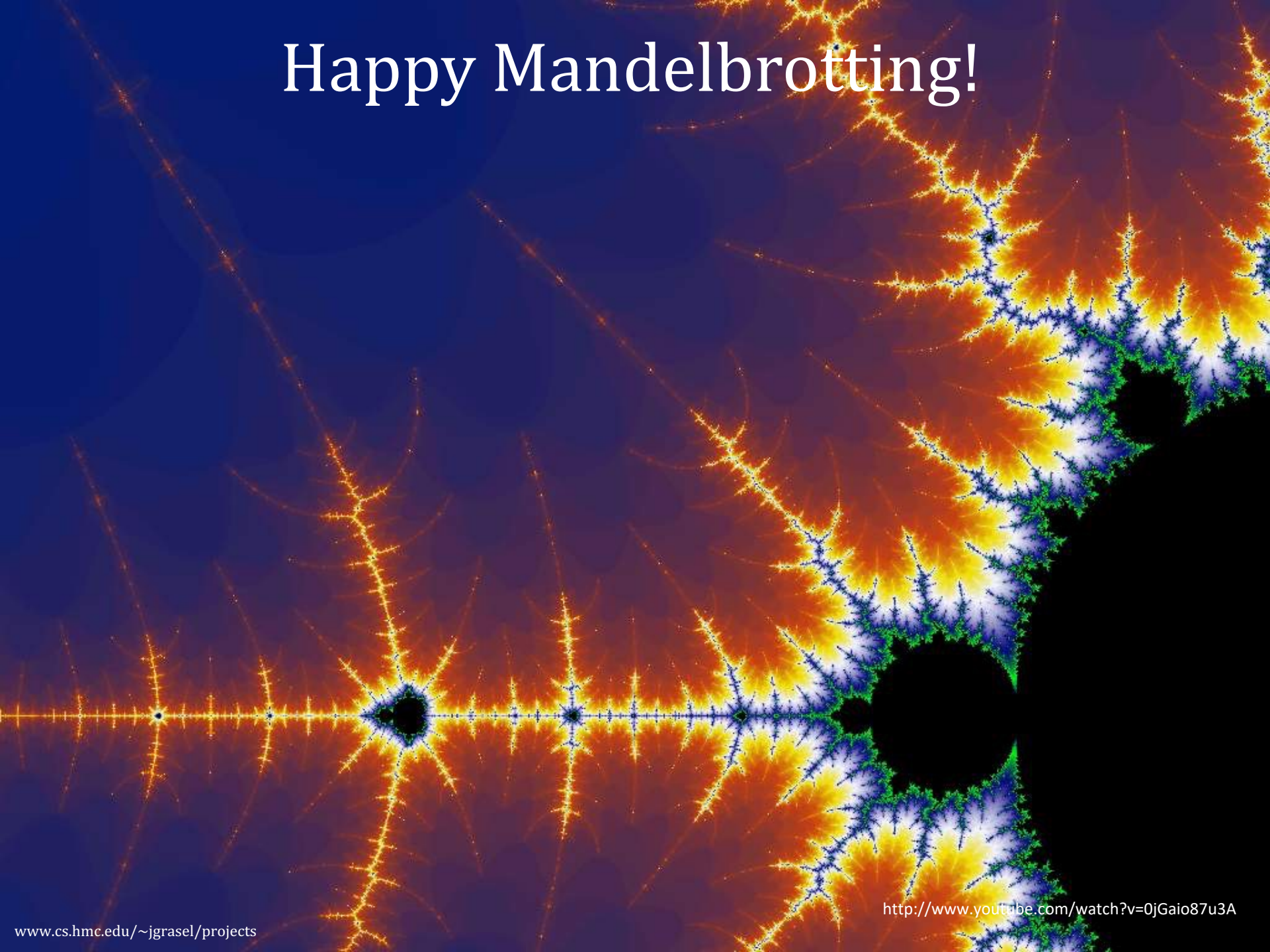


Numbers in yellow indicate the number of dendrites or spiral arms found in each region, and in the corresponding Julia fractals for each region.



In the *Seahorse Valley*....

Happy Mandelbrotting!



<http://www.youtube.com/watch?v=0jGaio87u3A>