Thinking in *loops*

for

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

list

while

```
x = 1
while x < 42:
    print(x)
    x += 1
```

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

Thinking in ***loops***

`for`	`while`
definite iteration	***indefinite iteration***
For a **known** list or # of iterations	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

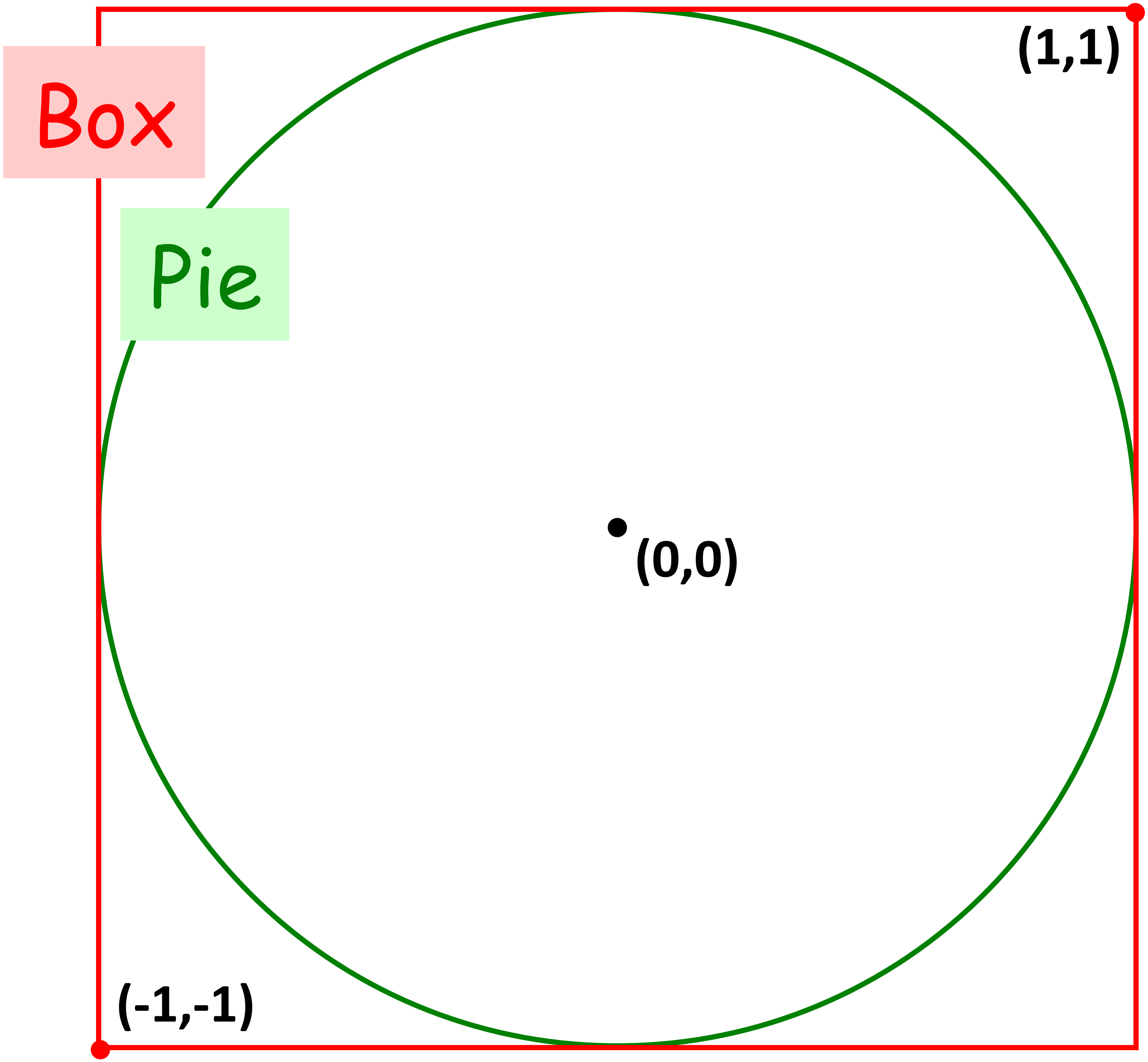
***Loopy* thinking**

Hw8 Pr3

Pi from Pie?

Pizza is the universal
constant, after all…

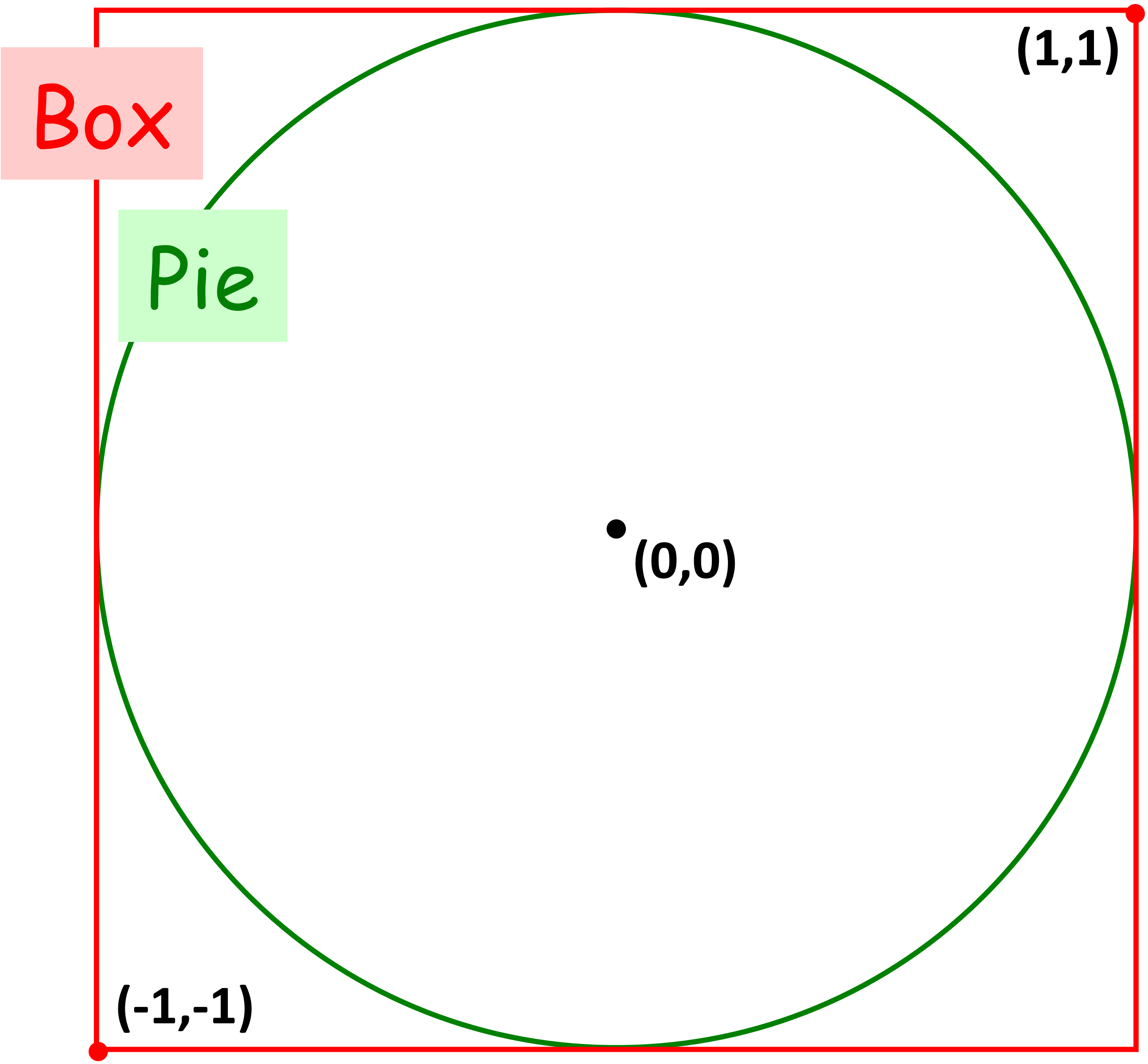
Hw8 Pr3



Estimating π from pie?

What if we just throw darts at this picture?

Pi-design challenge... *Name(s)* ______________________



Estimating π from pie?

(1) Suppose you throw 100 darts at the square (*all of them hit the square*)

(2) Suppose 80 of the 100 hit <u>inside</u> the circle.

(3) How could you estimate π from these throws?

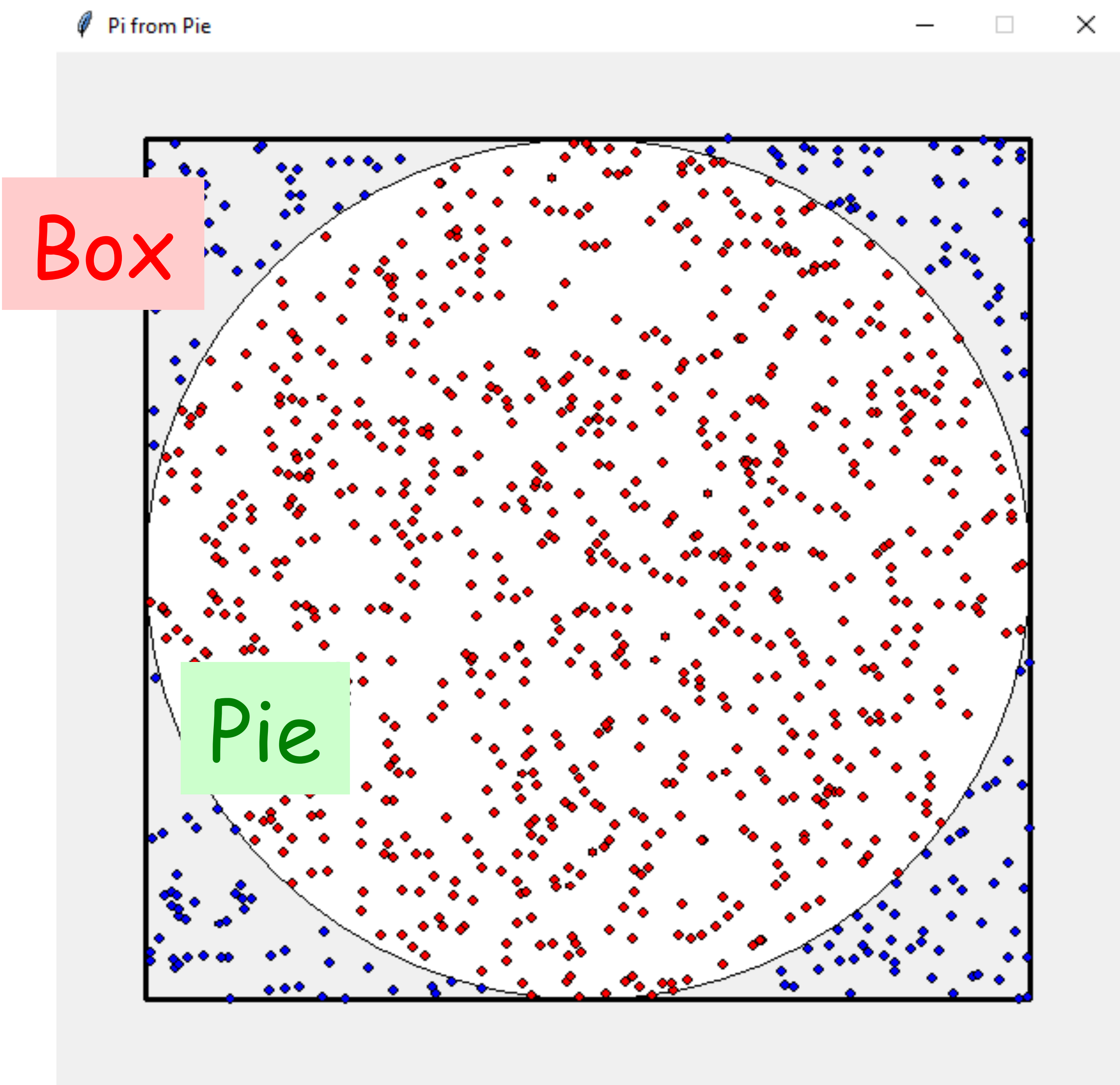
Hints

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)* ______________________



Estimating π from pie?

(1) Suppose you throw 100 darts at the square (*all of them hit the square*)

(2) Suppose 80 of the 100 hit <u>inside</u> the circle.

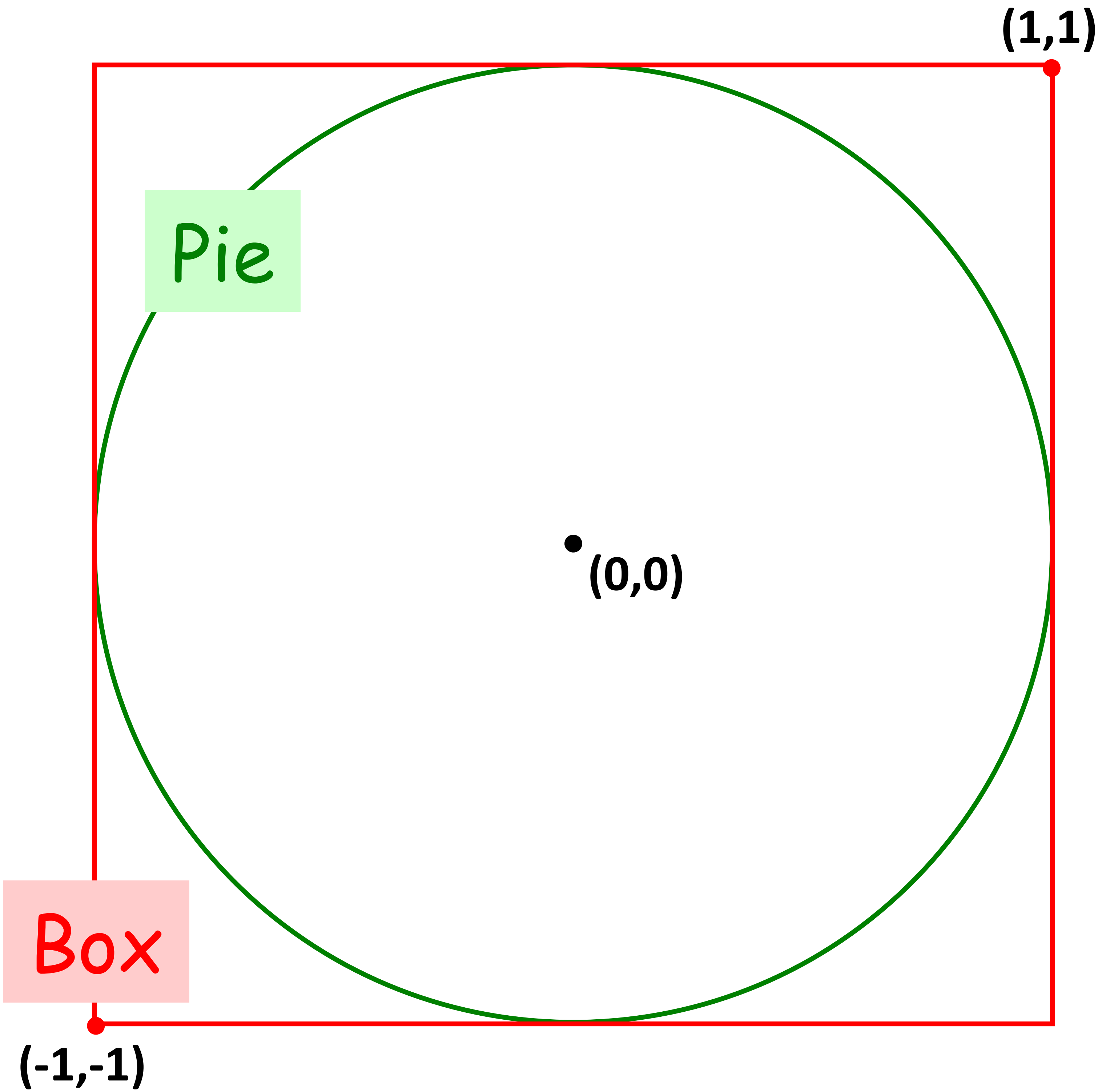
(3) How could you estimate π from these throws?

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!?



Hw8 Pr3

Estimating π from pie!

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

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

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

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

Not just loops...
Nested loops

Nested loops are familiar, too!

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

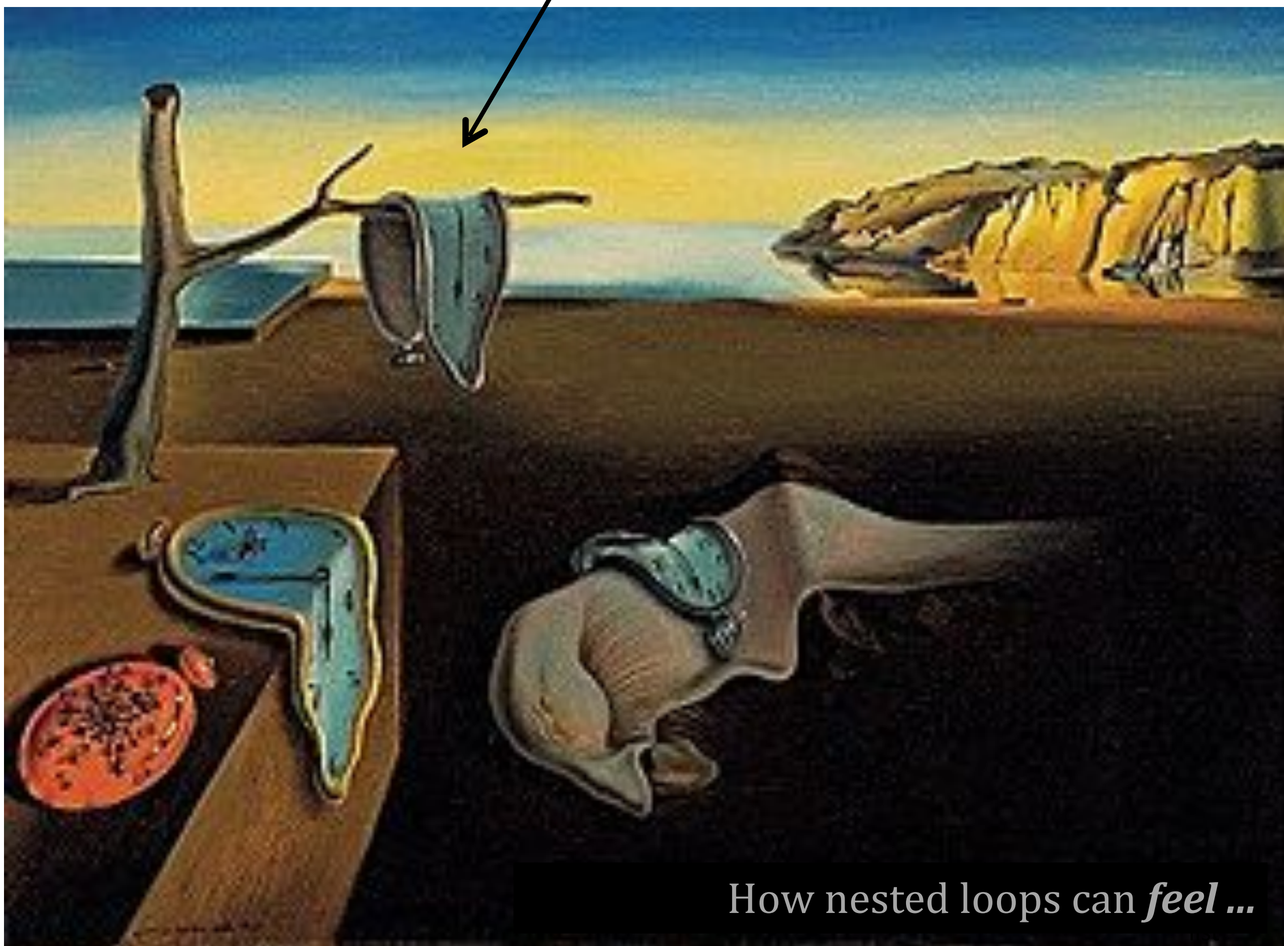
Nested loops are familiar, too!

Nested loops

Life clock


```
                list
for y in range(84):
                  list
  for m in range(12):
                      list
    for d in range(f(m,y)):
                        list
      for h in range(24):
                             list
        for mn in range(60):
                              list
          for s in range(60):
            tick()
```

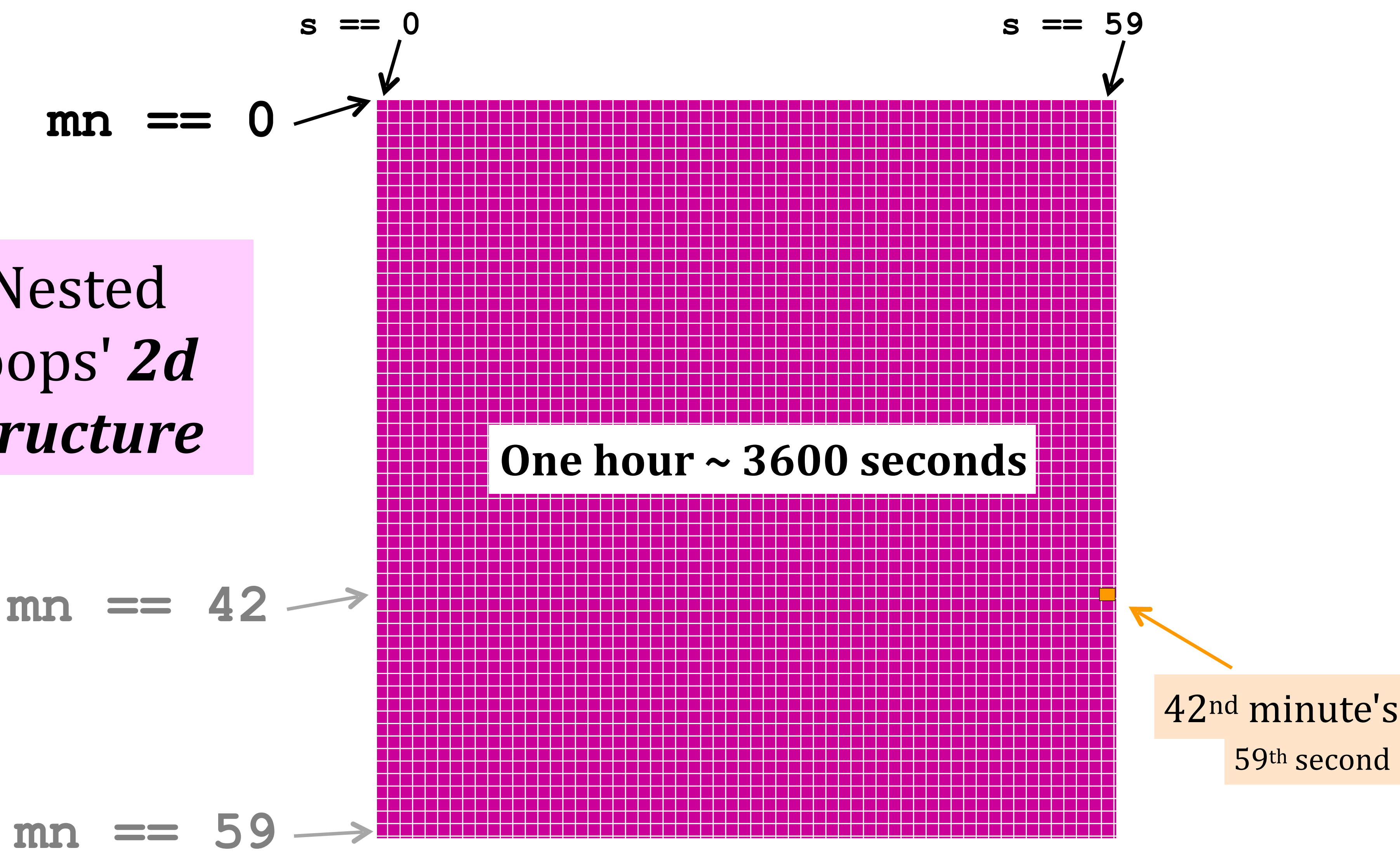
Nested loops!


How nested loops can *feel* ...

Persistence of Memory, S. Dali (MoMA)


```
:
f(m,y)):
e(24):
for m in range(60):
  for s in range(60):
    tick()
```

Nested loops' ***2d structure***




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

list
list

hour()

Creating 2d structure ~ in ASCII

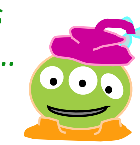
		col			
		0	1	2	3
row	0	#	#	#	#
	1	#	#	#	#
	2	#	#	#	#

list

list

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

Wait! this needs something more…



Creating 2d structure

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

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

Creating 2d structure

```
                       [0,1,2]
for row in list range(3):

        for col in list range(4):
            print('#',end='')

    print()
```

```
row =
    col =
    col =
    col =
    col =

row =
    col =
    col =
    col =
    col =

row =
    col =
    col =
    col =
    col =
```

		col		
	0	1	2	3
row 0				
row 1				
row 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()
```

list list

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 code creates the fourth one?

* and ** are extra!

Name(s): ______________________

Try it!

A

[0,1,2]

[0,1,2,3,4,5]

```
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

rows \ cols	0	1	2	3	4	5
0	#	#	#	#		
1	#	#	#			
2	#	#				

2

rows \ cols	0	1	2	3	4	5
0	#		#		#	
1		#		#		#
2	#		#		#	

*****

rows \ cols	0	1	2	3	4
0	#	#	#	#	#
1		#	#	#	
2			#		

3

rows \ cols	0	1	2	3	4	5
0		#	#	#	#	#
1			#	#	#	#
2				#	#	#

4

rows \ cols	0	1	2	3	4	5
0		#		#		#
1		#		#		#
2		#		#		#

******

rows \ cols	0	1	2	3	4
0		#	#	#	
1	#		#		#
2	#	#		#	#

Match!

What code creates the fourth one?

* and ** are extra!

Try this on the back first...

Quiz

A

[0,1,2]

[0,1,2,3,4,5]

```
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

rows \ cols	0	1	2	3	4	5
0	#	#	#	#		
1	#	#	#			
2	#	#				

`if c+r<4`

2

rows \ cols	0	1	2	3	4	5
0	#		#		#	
1		#		#		#
2	#		#		#	

*****

rows \ cols	0	1	2	3	4
0	#	#	#	#	#
1		#	#	#	
2			#		

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

3

rows \ cols	0	1	2	3	4	5
0		#	#	#	#	#
1			#	#	#	#
2				#	#	#

4

rows \ cols	0	1	2	3	4	5
0		#		#		#
1		#		#		#
2		#		#		#

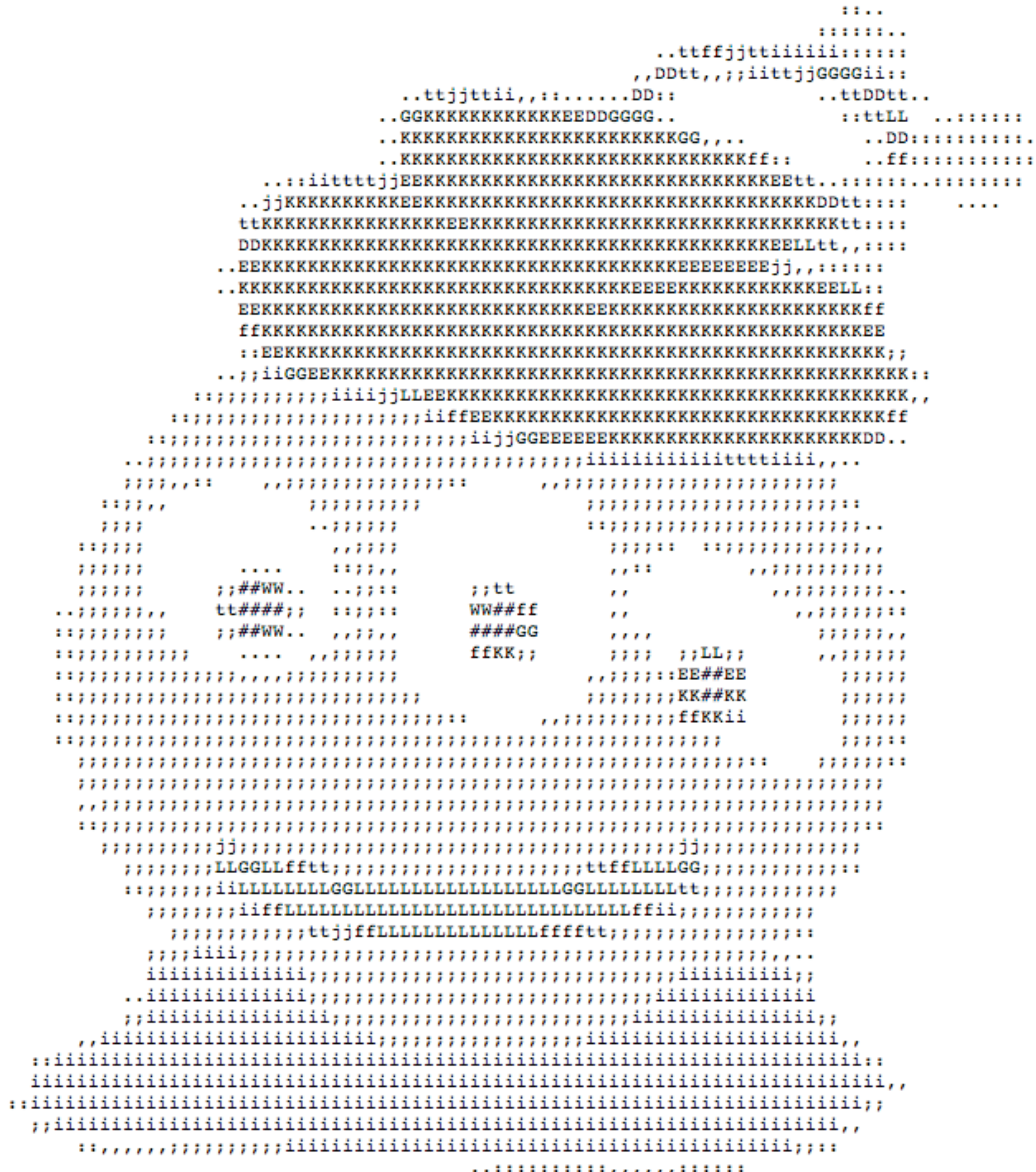
******

rows \ cols	0	1	2	3	4
0		#	#	#	
1	#		#		#
2	#	#		#	#

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

Nested loops: from ASCII Art

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



... to *"real"* images!

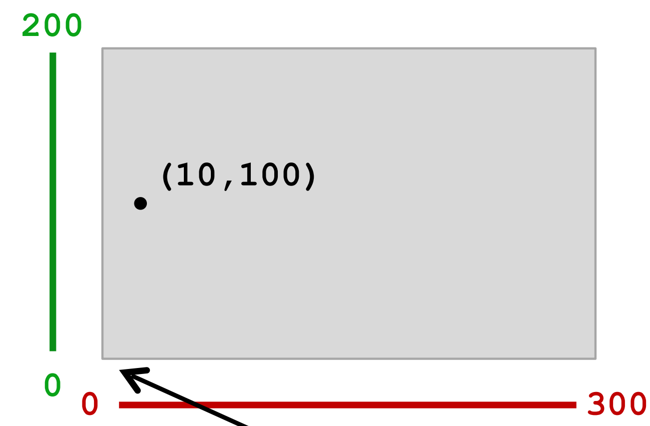
Python and images

```
from cs5png import *
```

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

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

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



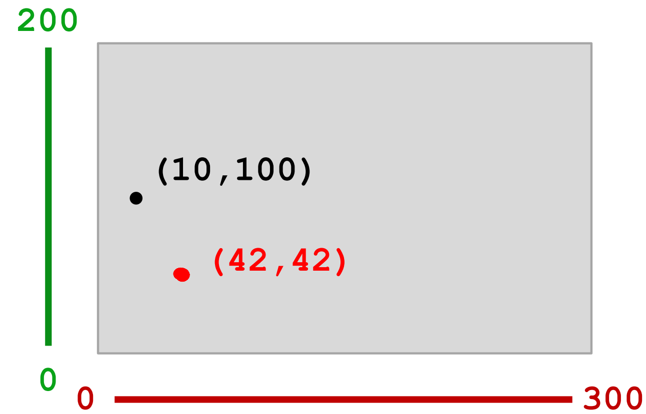
(0,0) is in the usual place!

Python and images

```
from cs5png import *
```

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

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



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

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

Imagining Images

```
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()
```

thicker line?

other diagonal?

stripes ?

thicker stripes?

thatching?

Complex #s !

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

Nothing's too *complex for Python!*

i *can't believe this!*

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

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

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

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

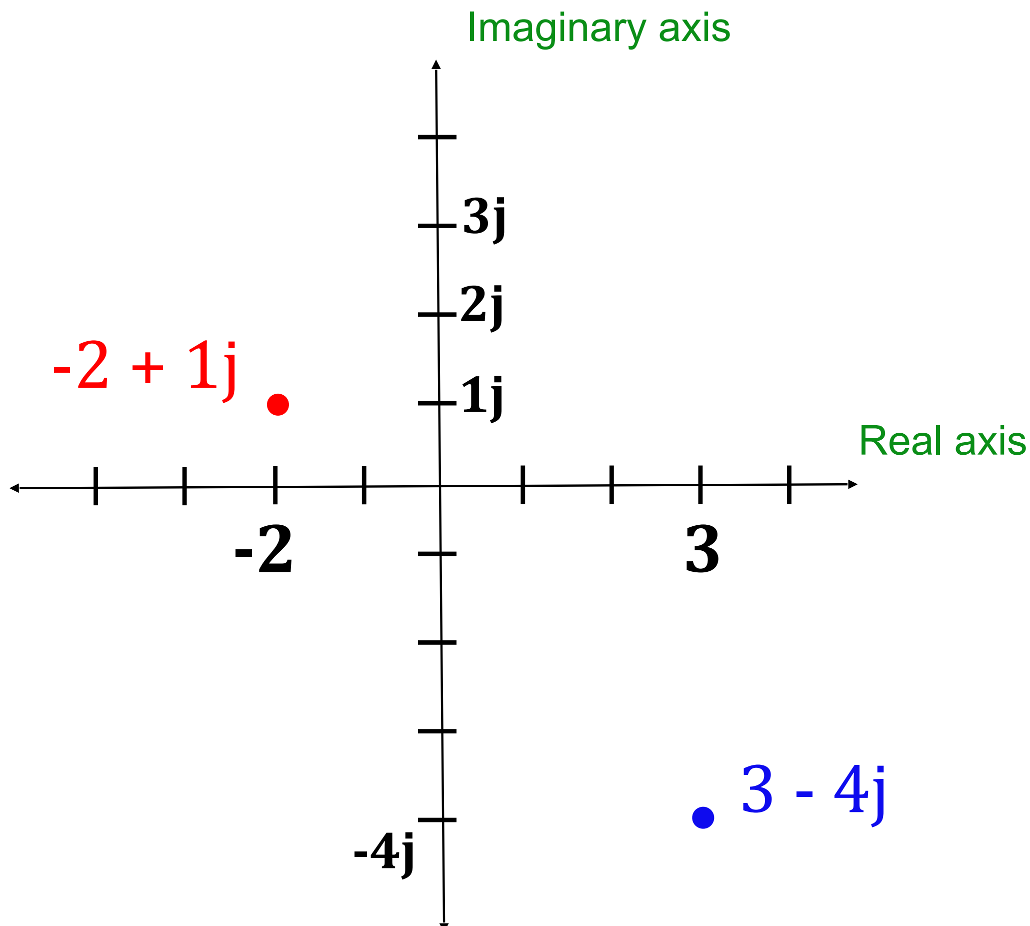
Complex #s !

Nothing's too ***complex for Python!***

i *can't believe this!*

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

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

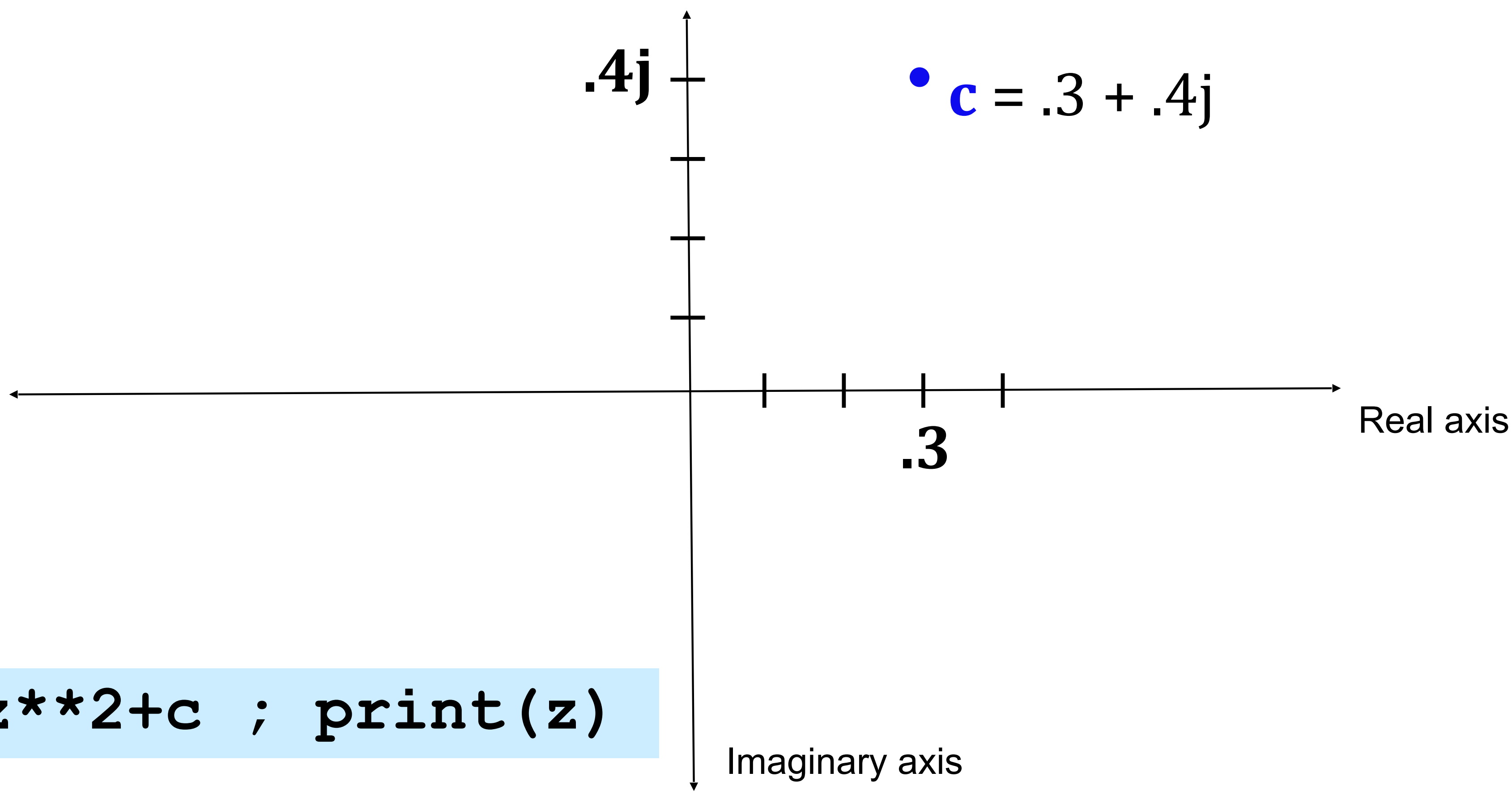


Lab 8: *the Mandelbrot Set*

Consider an ***update rule***
for all complex numbers ***c***

$$\mathbf{z_0 = 0}$$

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




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

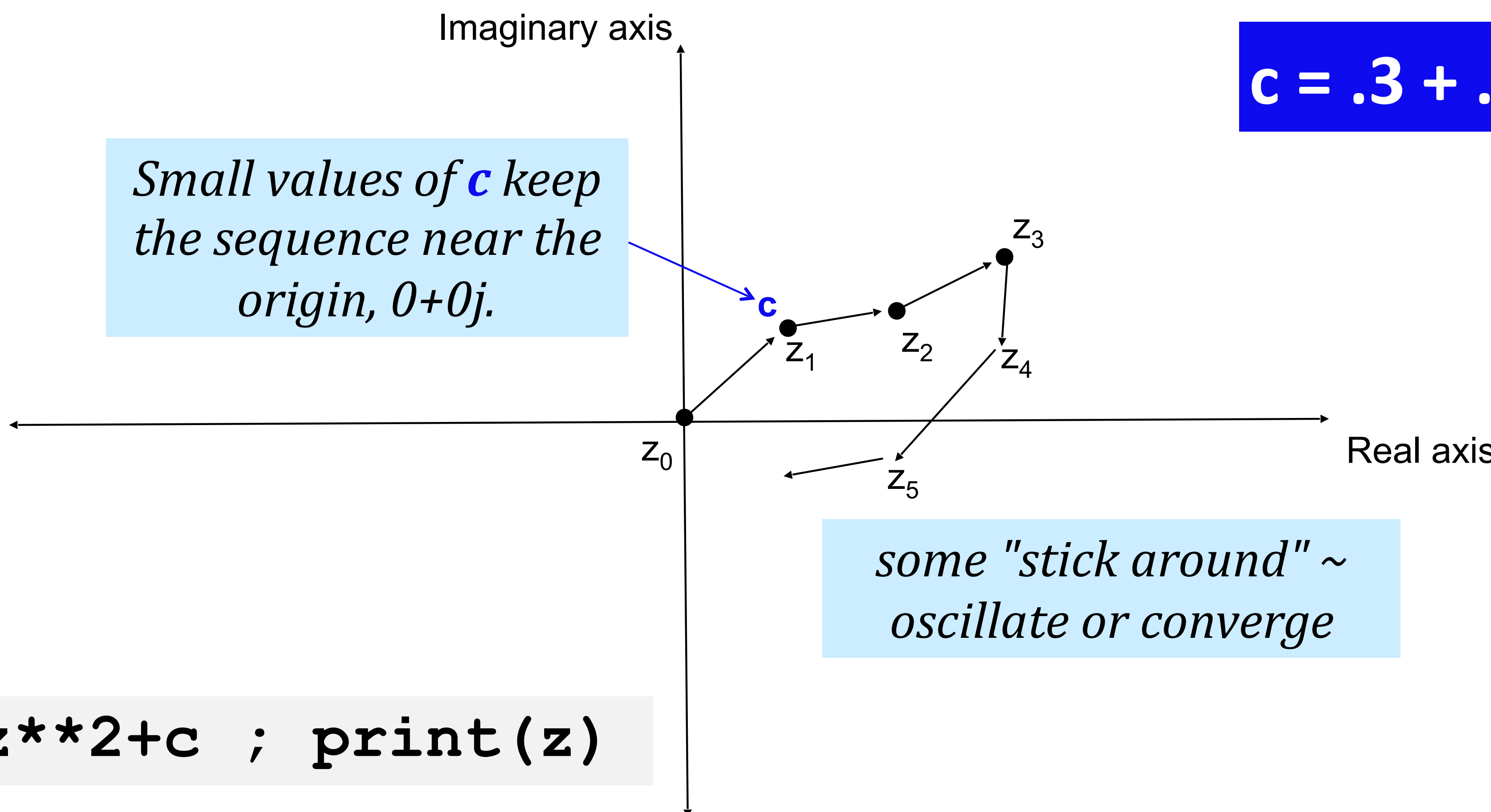
Mandelbrot Definition

Consider an ***update rule***
for all complex numbers ***c***

$$\mathbf{z_0 = 0}$$

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

c = .3 + .4j



*Small values of **c** keep the sequence near the origin, 0+0j.*

some "stick around" ~ oscillate or converge

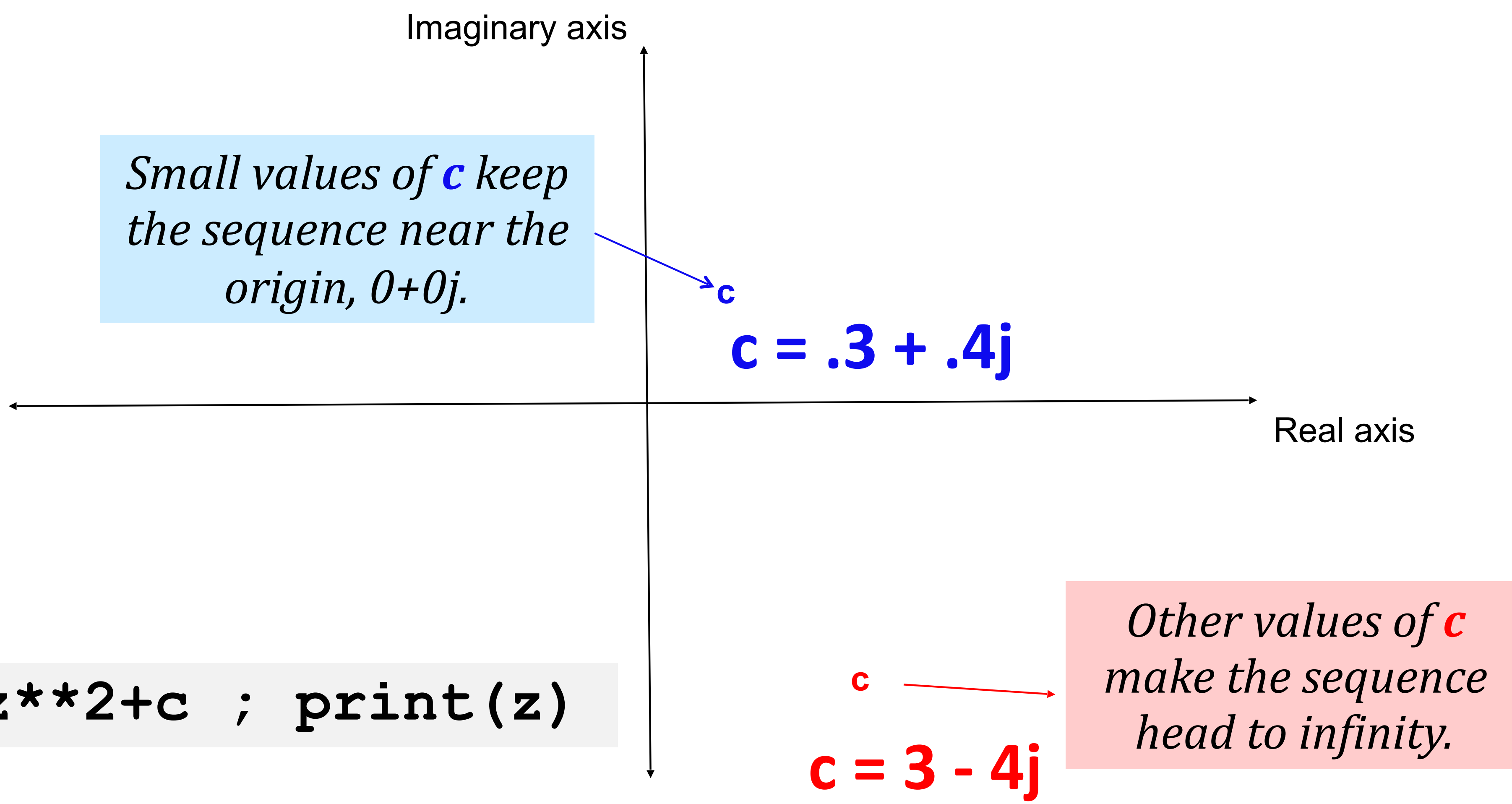
```
z = z**2+c ; print(z)
```

Mandelbrot Definition

Consider an ***update rule***
for all complex numbers ***c***

$$z_0 = 0$$

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



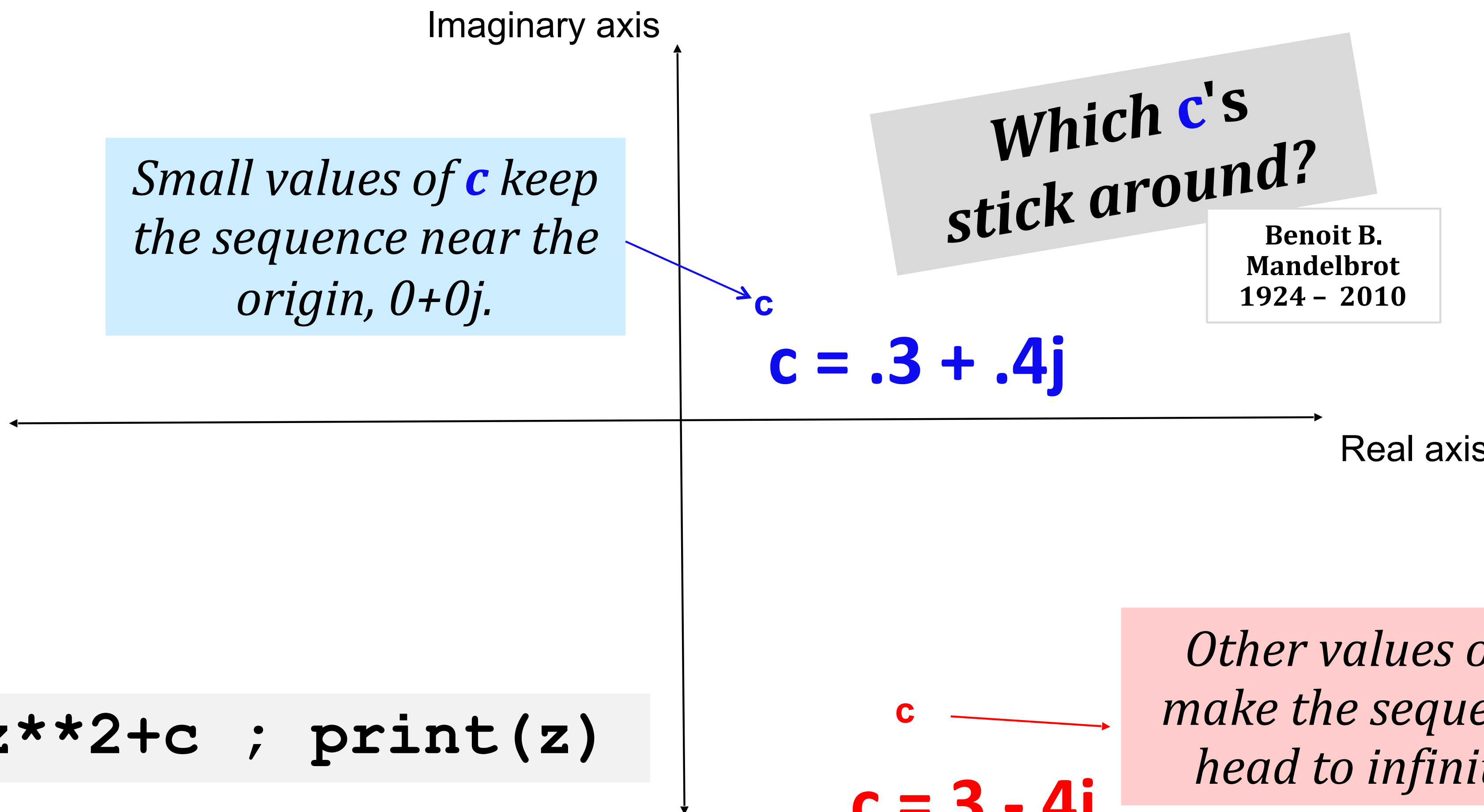

```
z = z**2+c ; print(z)
```

Mandelbrot Definition

Consider an ***update rule***
for all complex numbers ***c***

$$z_0 = 0$$

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



*Small values of **c** keep the sequence near the origin, 0+0j.*

Which c's stick around?

Benoit B. Mandelbrot 1924 – 2010

*Other values of **c** make the sequence head to infinity.*

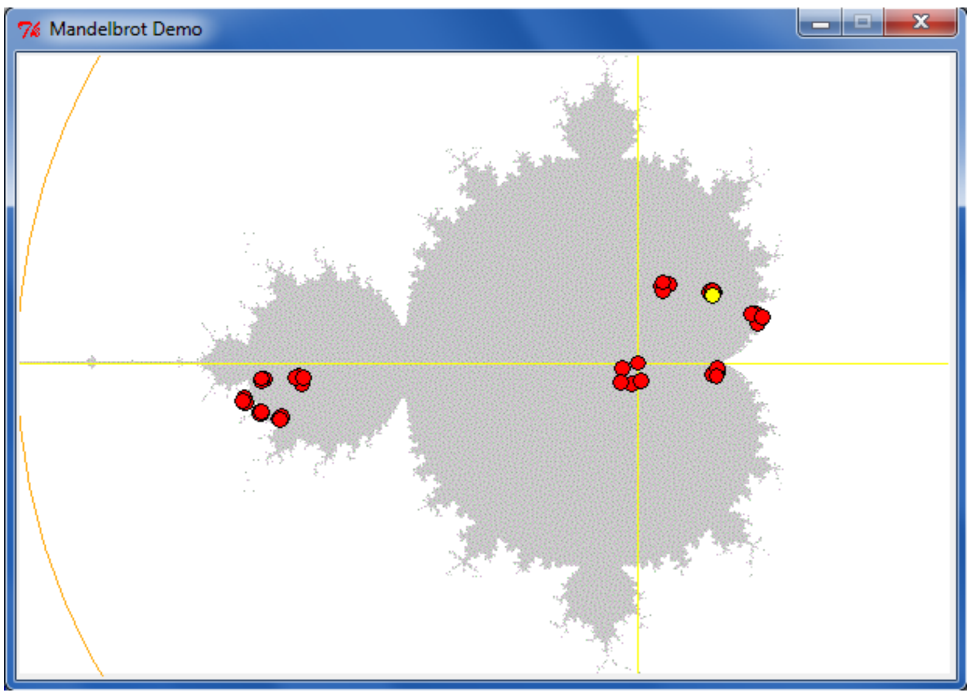
```
z = z**2+c ; print(z)
```

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 + -0.054634646374 * 1j
iter # 4 : z = 0.256047527535 + -0.030302692070 * 1j
iter # 5 : z = -1.14904739926 + -0.17842 6935 * 1j
iter # 6 : z = 0.0747849173552 7964 * 1j
iter # 7 : z = -1.26917 5977 * 1j
iter # 8 : z 88 * 1j
iter # 9 08 * 1j
iter # 10 56431 * 1j
iter # 11 602 * 1j
iter # 12 14 * 1j
iter # 13 5207 * 1j
iter # 14 : 27677689 * 1j
iter # 15 : + -0.180471770237 * 1j
iter # 16 : 360076381 + 0.255350697358 * 1j
iter # 17 : = -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
```

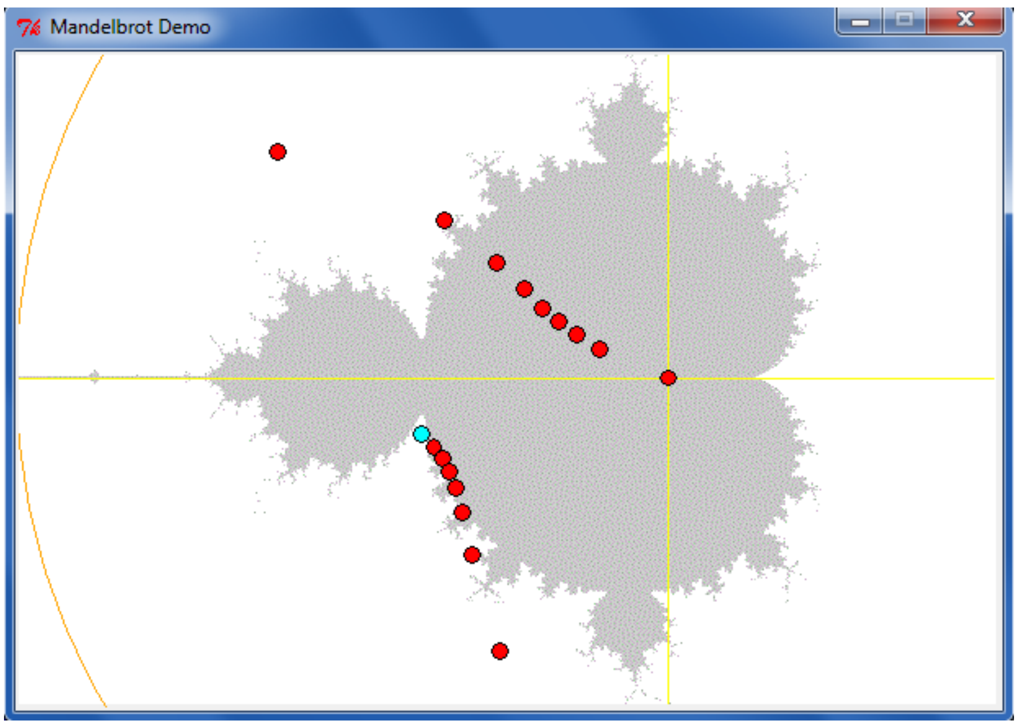
some ***c***'s stick around

Lab 8: *the Mandelbrot Set*

Consider an ***update rule***
for all complex numbers ***c***

$$z_0 = 0$$

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



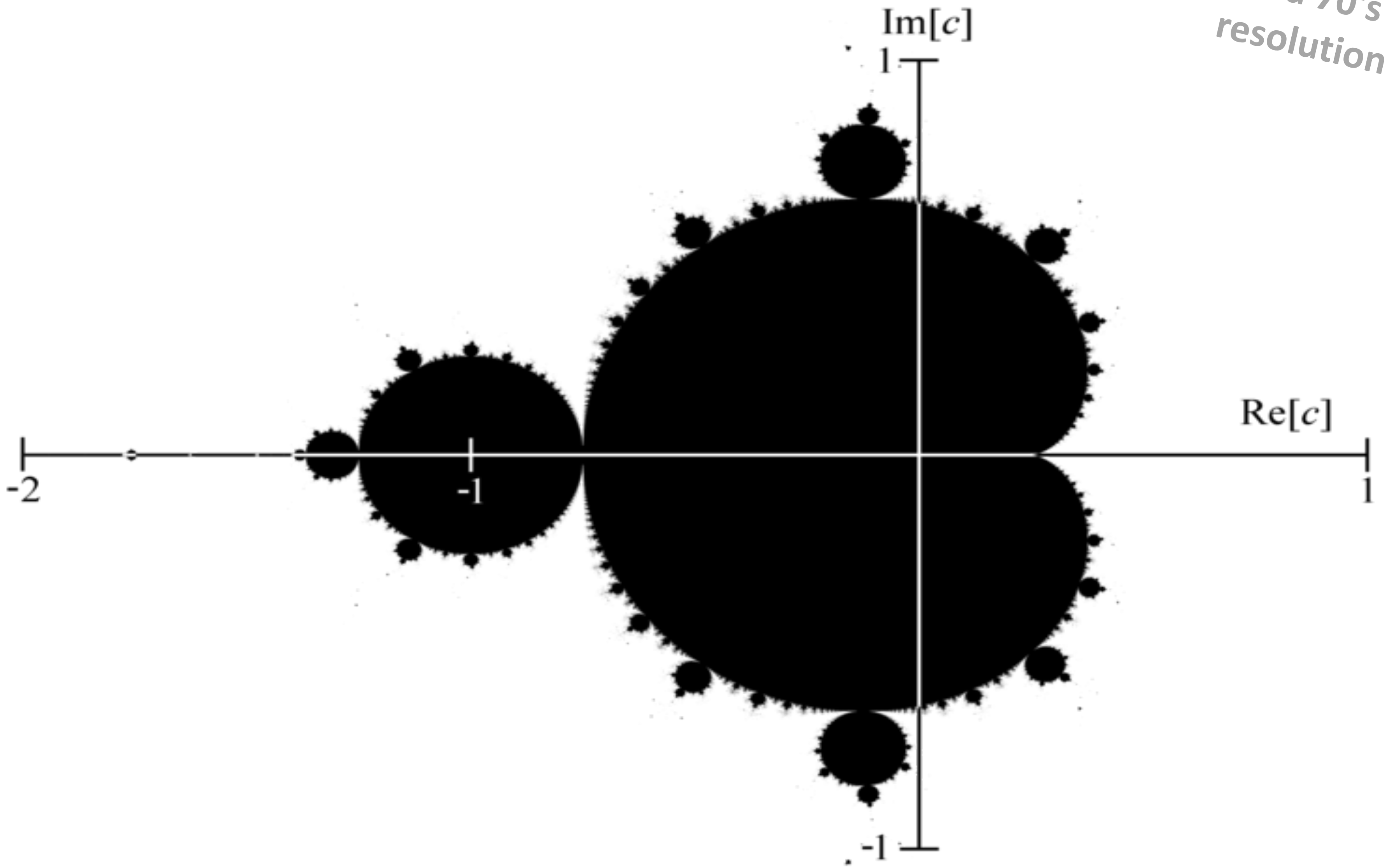

```
Click to choose c.
c is -0.757929883139 + -0.172932330827 * 1j

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 + 0.1310   537188 * 1j
iter # 5 : z = -0.69471444654          369601 * 1j
iter # 6 : z = -0.336                   34238 * 1j
iter # 7 :                              197352 * 1j
iter # 8 :                              9922 * 1j
iter # 9 :                              50516 * 1j
iter # 10 :                             34007 * 1j
iter # 11 :                             0519 * 1j
iter # 12 :                             438 * 1j
iter # 13 :                   43792492805 * 1j
iter # 14 : z                  018 + 0.48336747636 * 1j
iter # 15 : z    16174815348 + -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

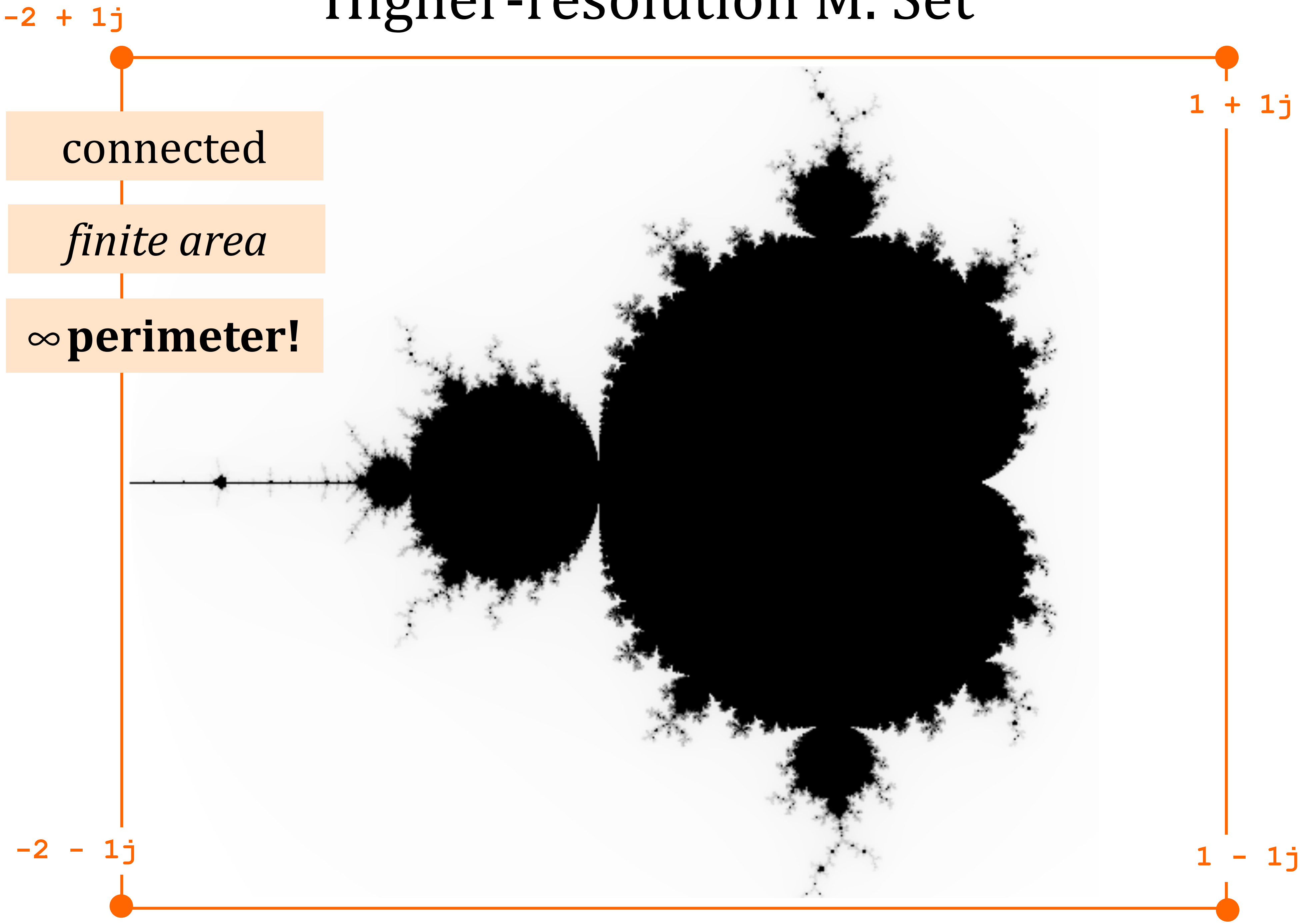
Mandelbrot Set ~ *points that stick around*

mid 70's resolution



The shaded area are points that do ***not*** diverge for `z = z**2 + c`

Higher-resolution M. Set



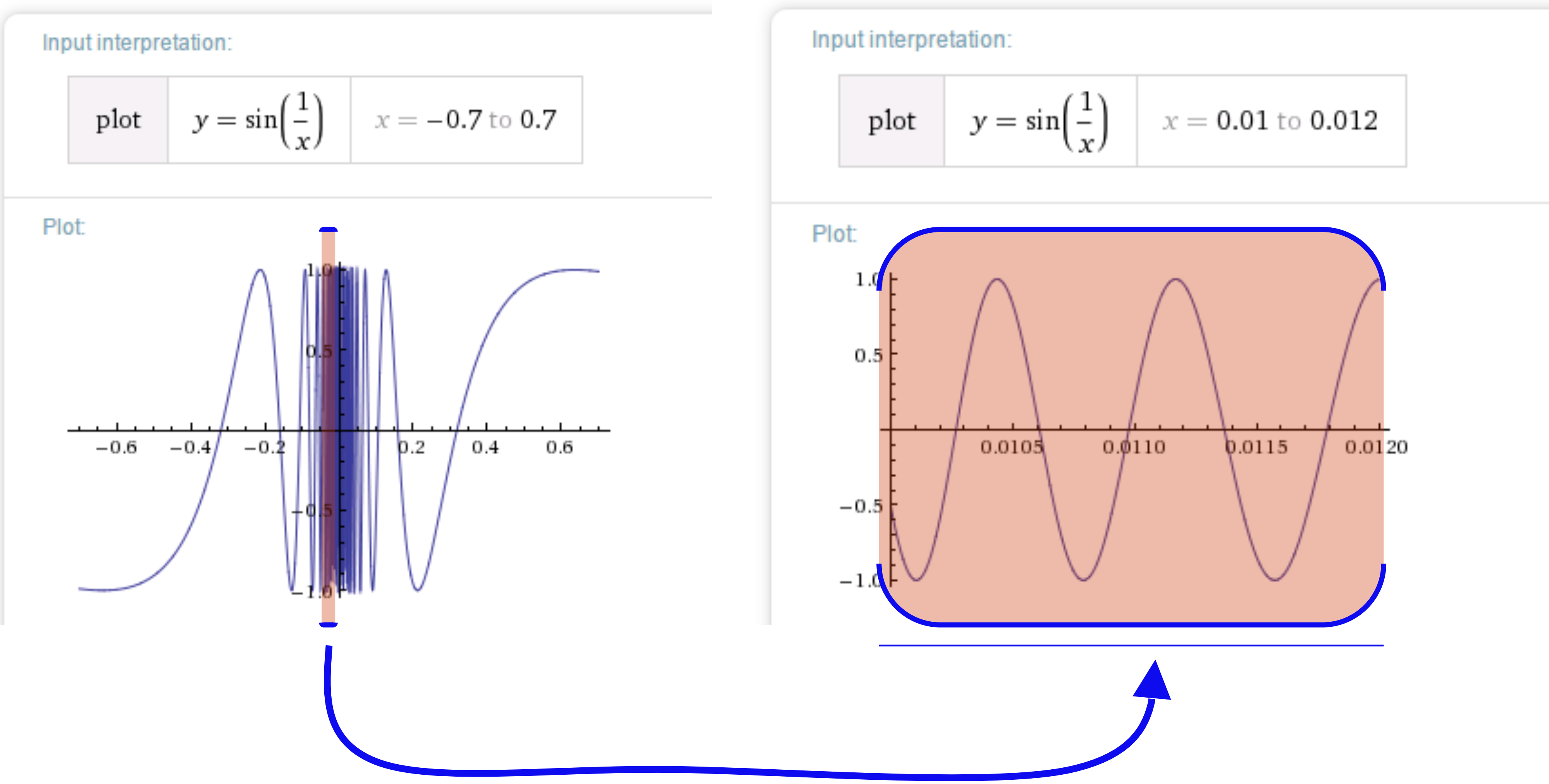
connected

finite area

∞ **perimeter!**

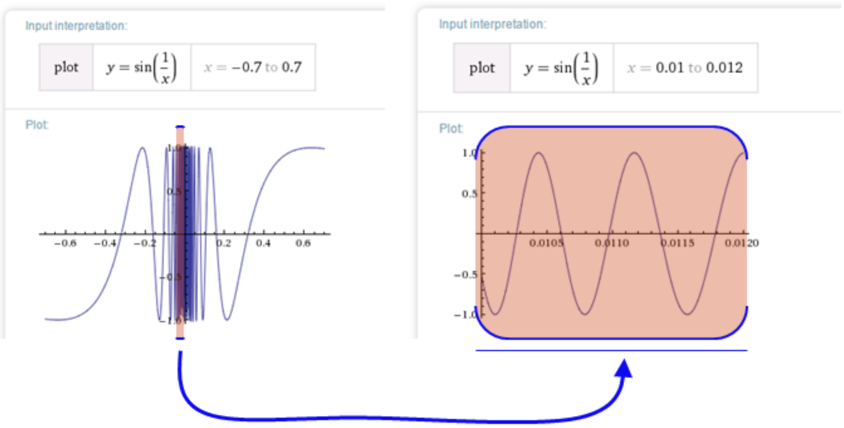
The black pixels are points that do ***not*** diverge for `z = z**2 + c`

Chaos?



Complex things always consisted of simple parts…

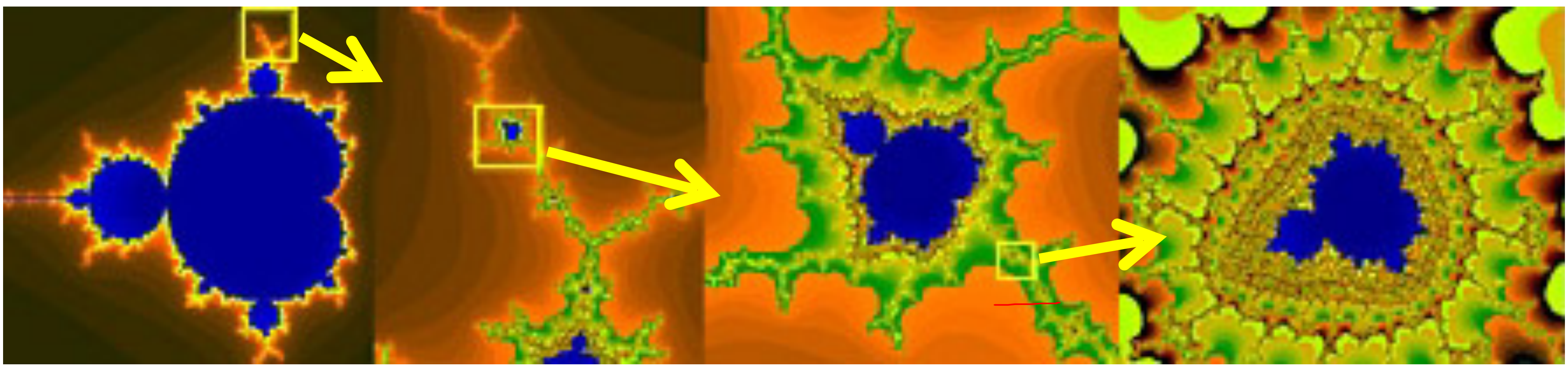
Before the M. Set, complex things
were made of simple parts:



Chaos!

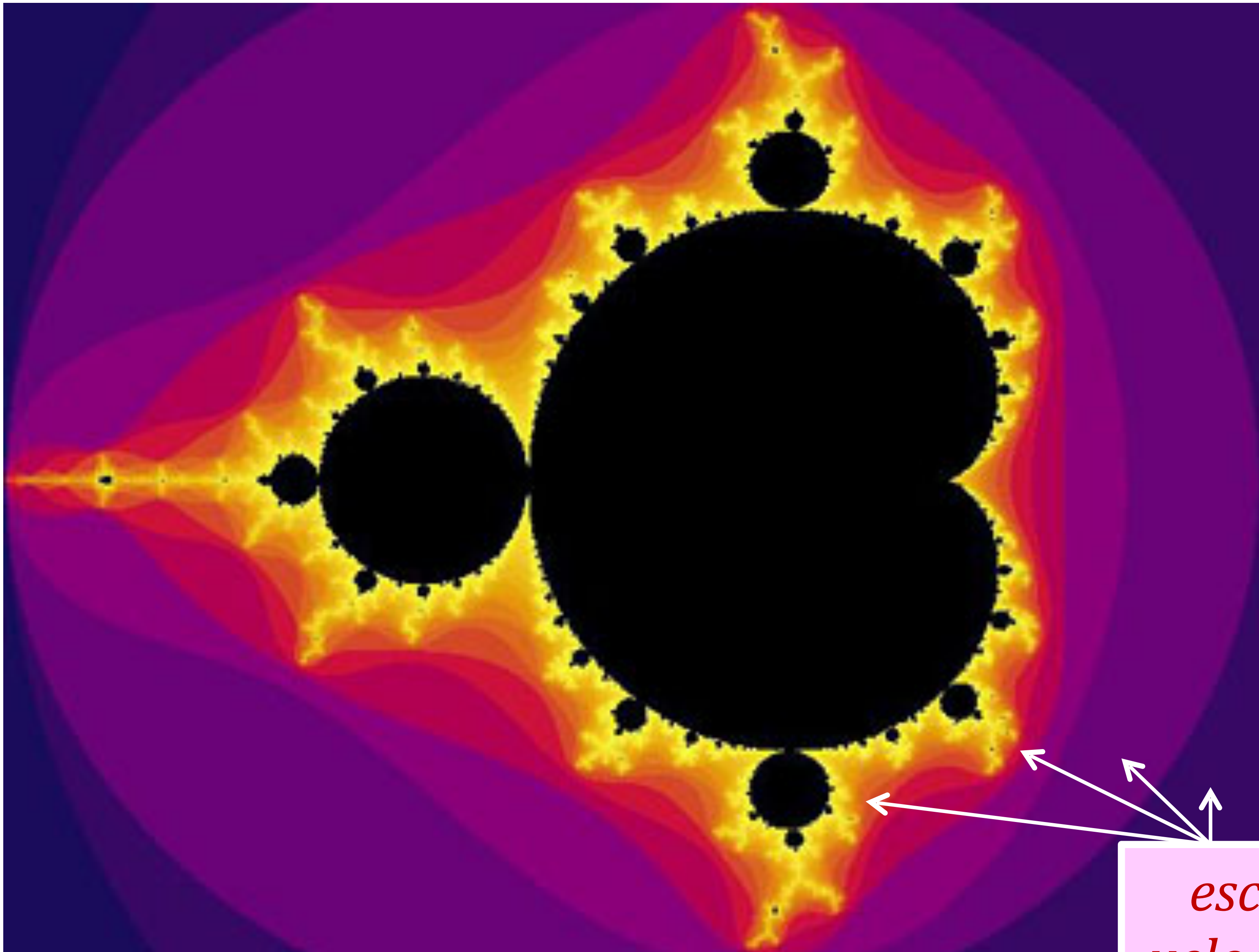
This was a *"naturally occurring"* object where
zooming uncovers ***more*** detail, not less:

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



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

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

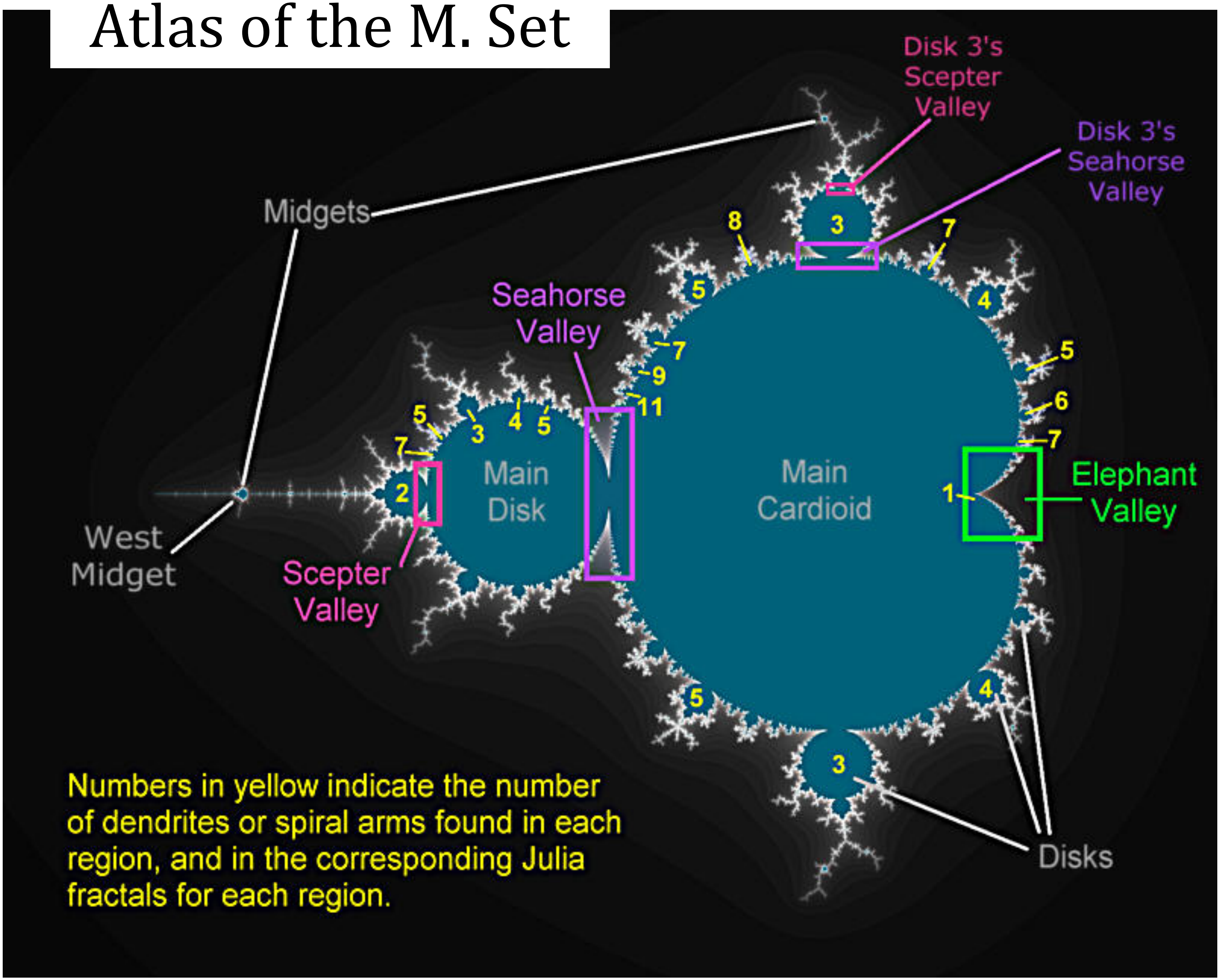


escape velocities!

What are these colors?

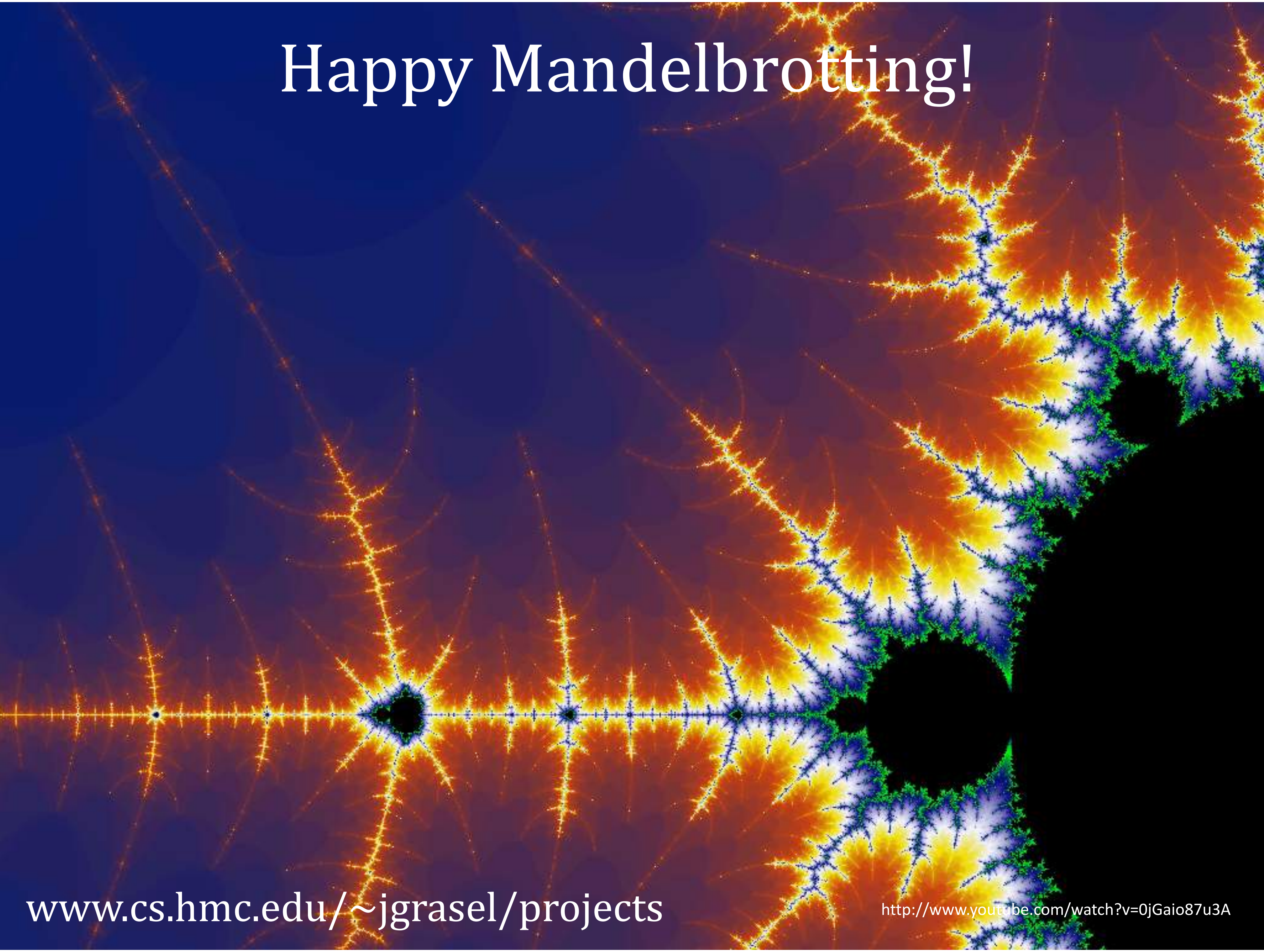
??

Atlas of the M. Set



In the *Seahorse Valley*….

Happy Mandelbrotting!



www.cs.hmc.edu/~jgrasel/projects

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