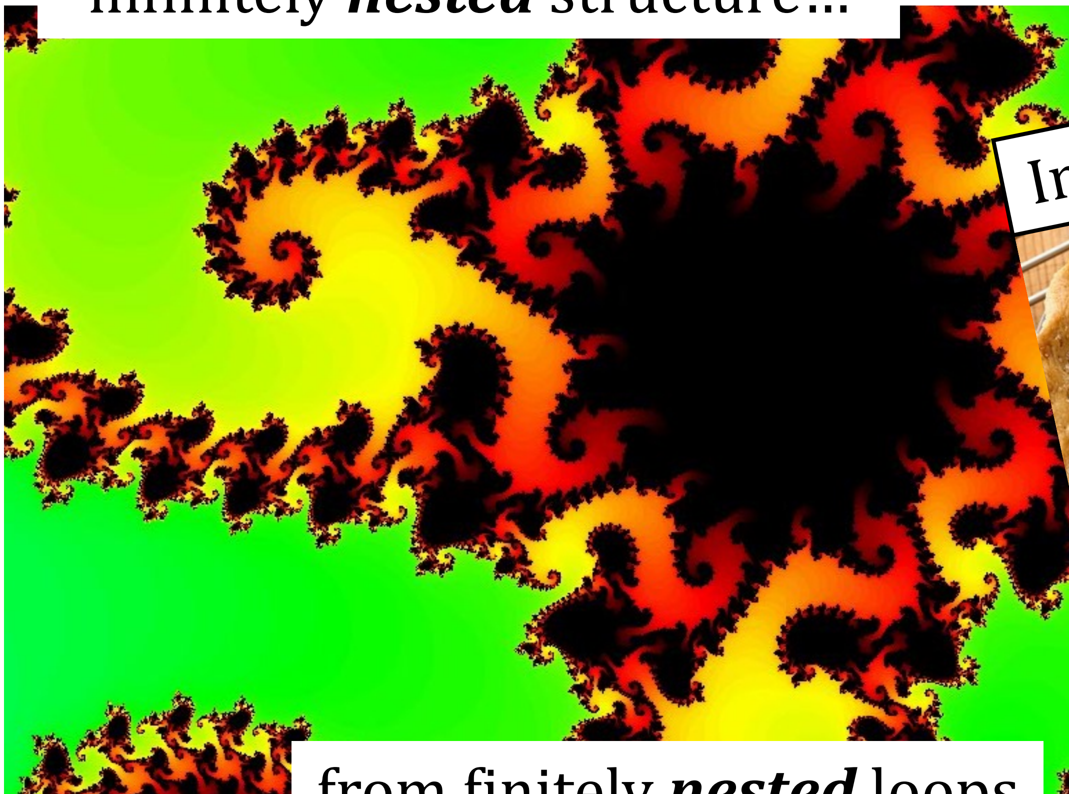
Piece of pie bud?

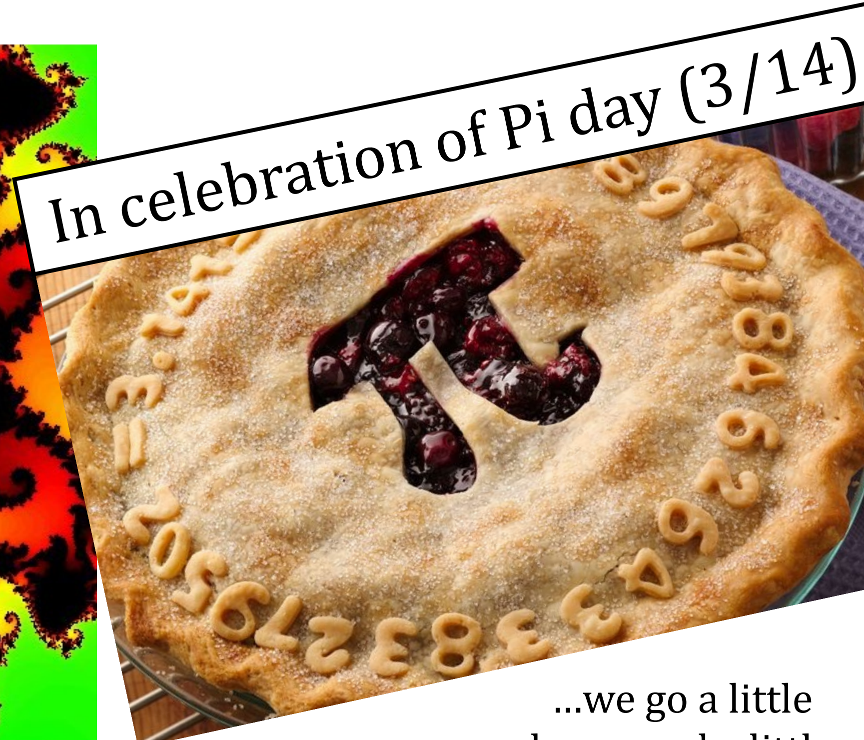
CS 5 Today

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

list

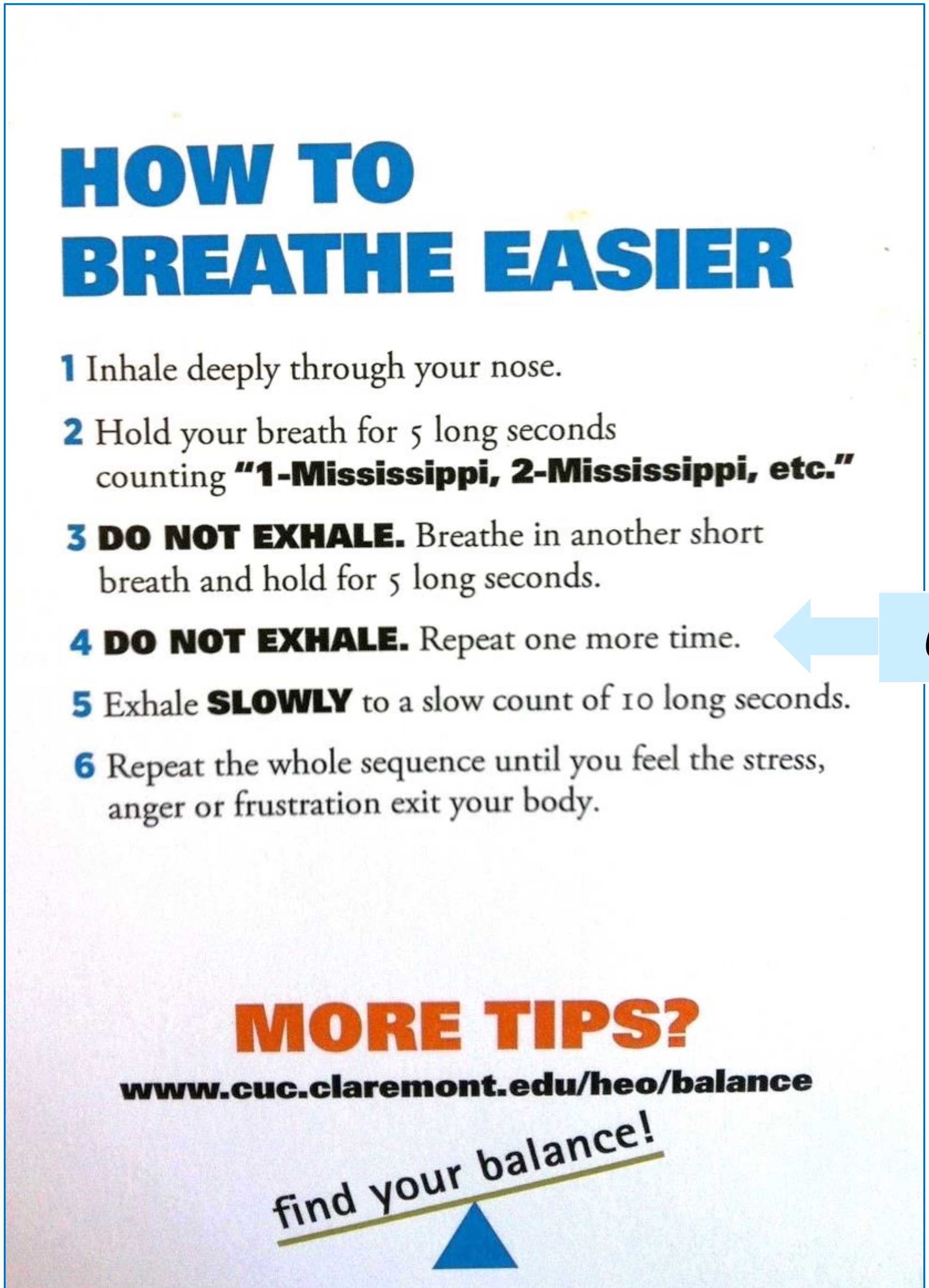
```
for x in range(42):
    print(x)
```

while

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

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

Loop design...

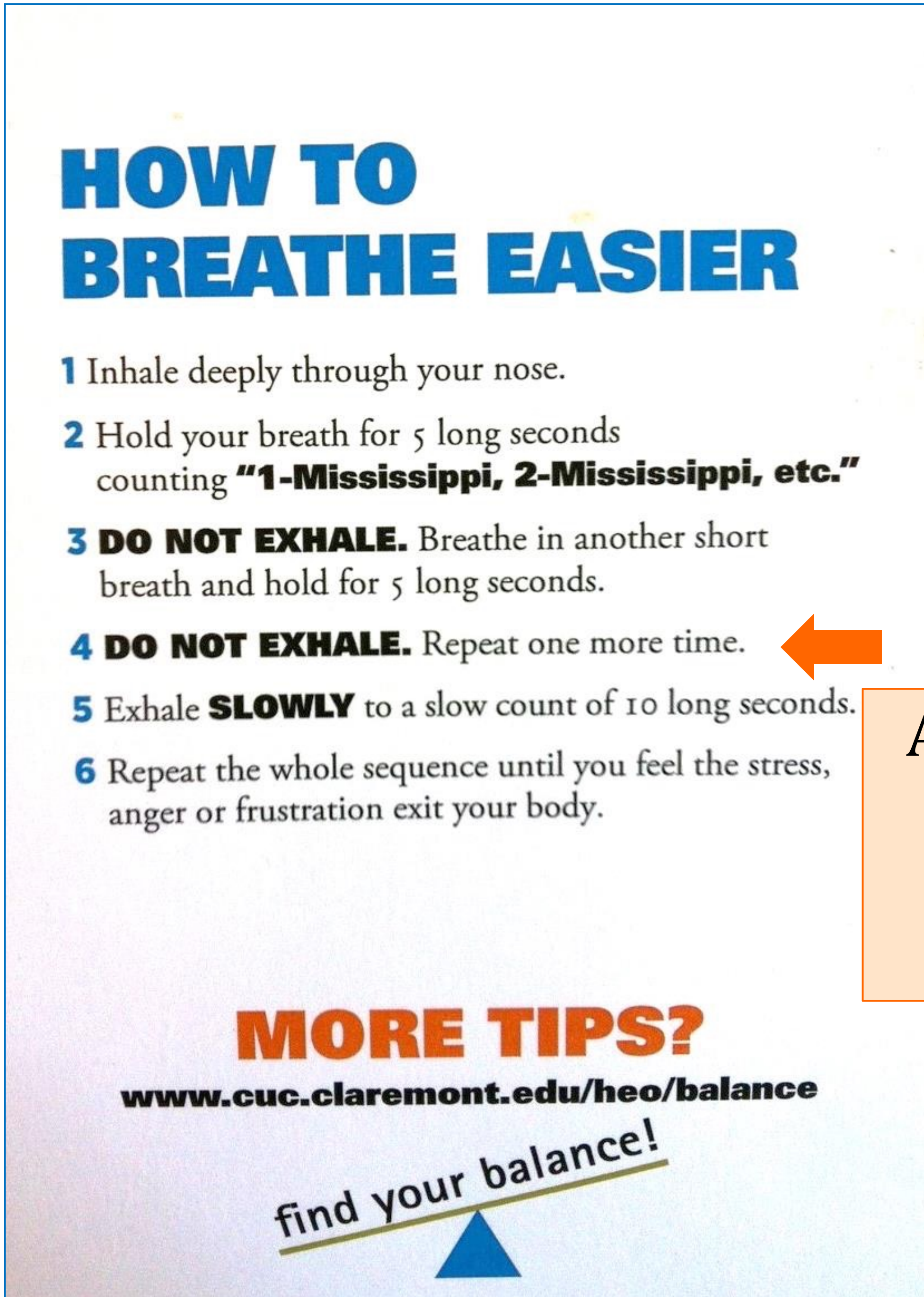


Careful here!

Is this a `for` or a `while` loop?

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

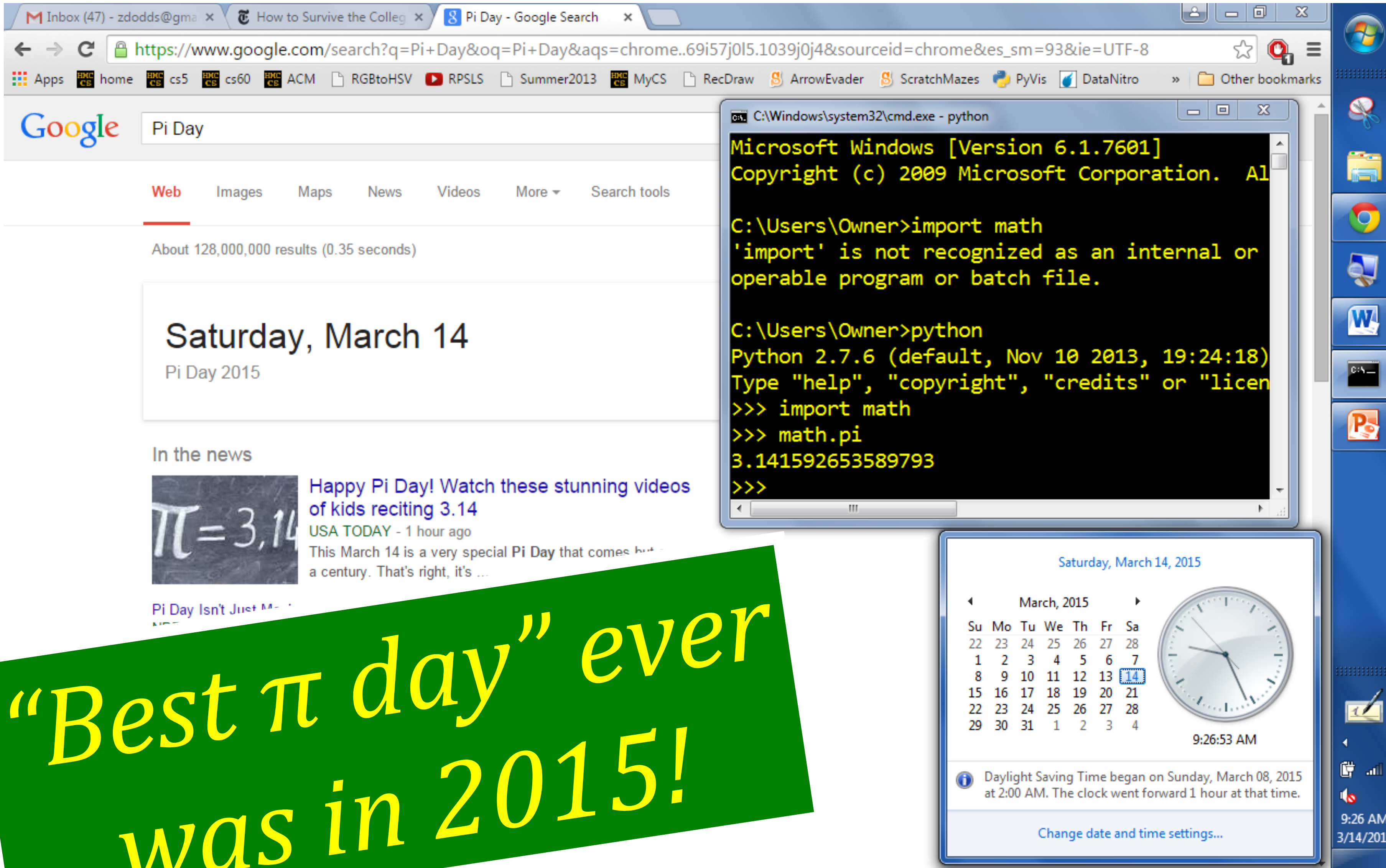
Homework 8 preview

#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

π day!

3/14/15 9:26:53



"Best π day" ever was in 2015!

Hw8 Pr3

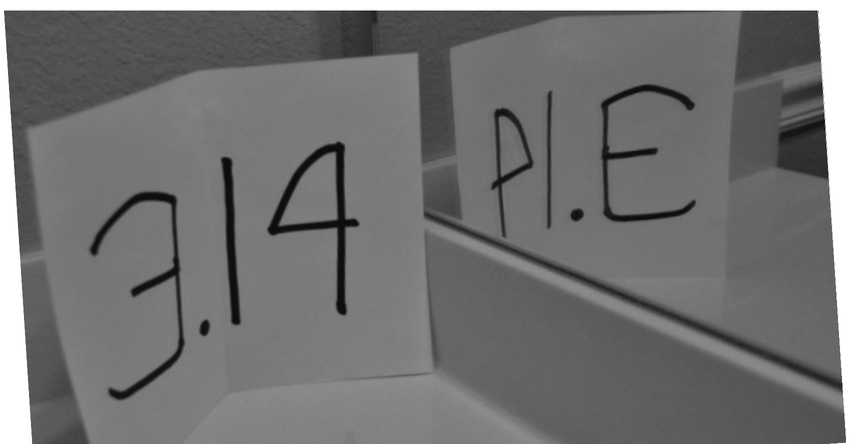
Honoring "π day"

π from Pie?

Box

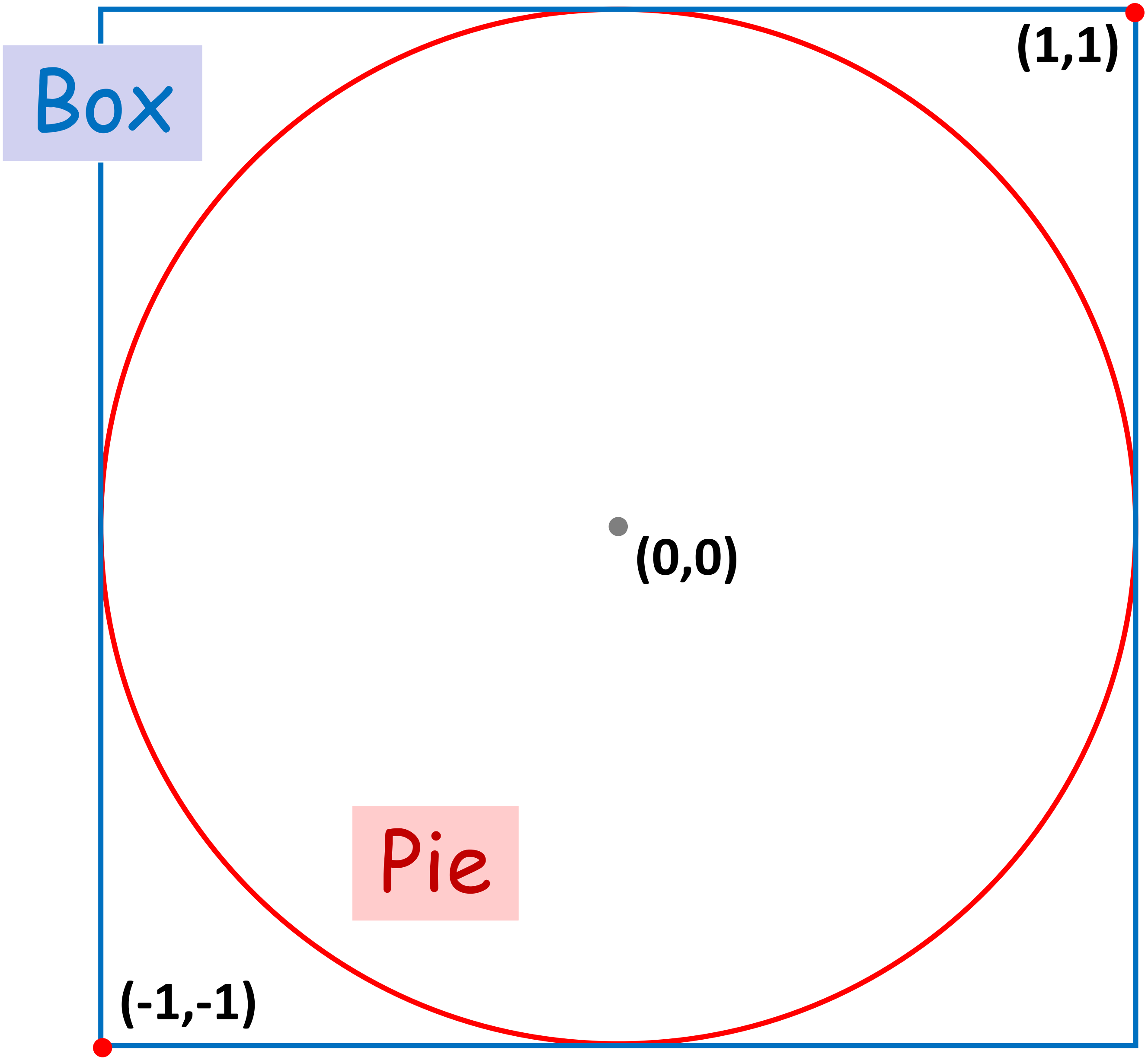
Pie

π



This *couldn't* be just a coincidence!

Hw8 Pr3

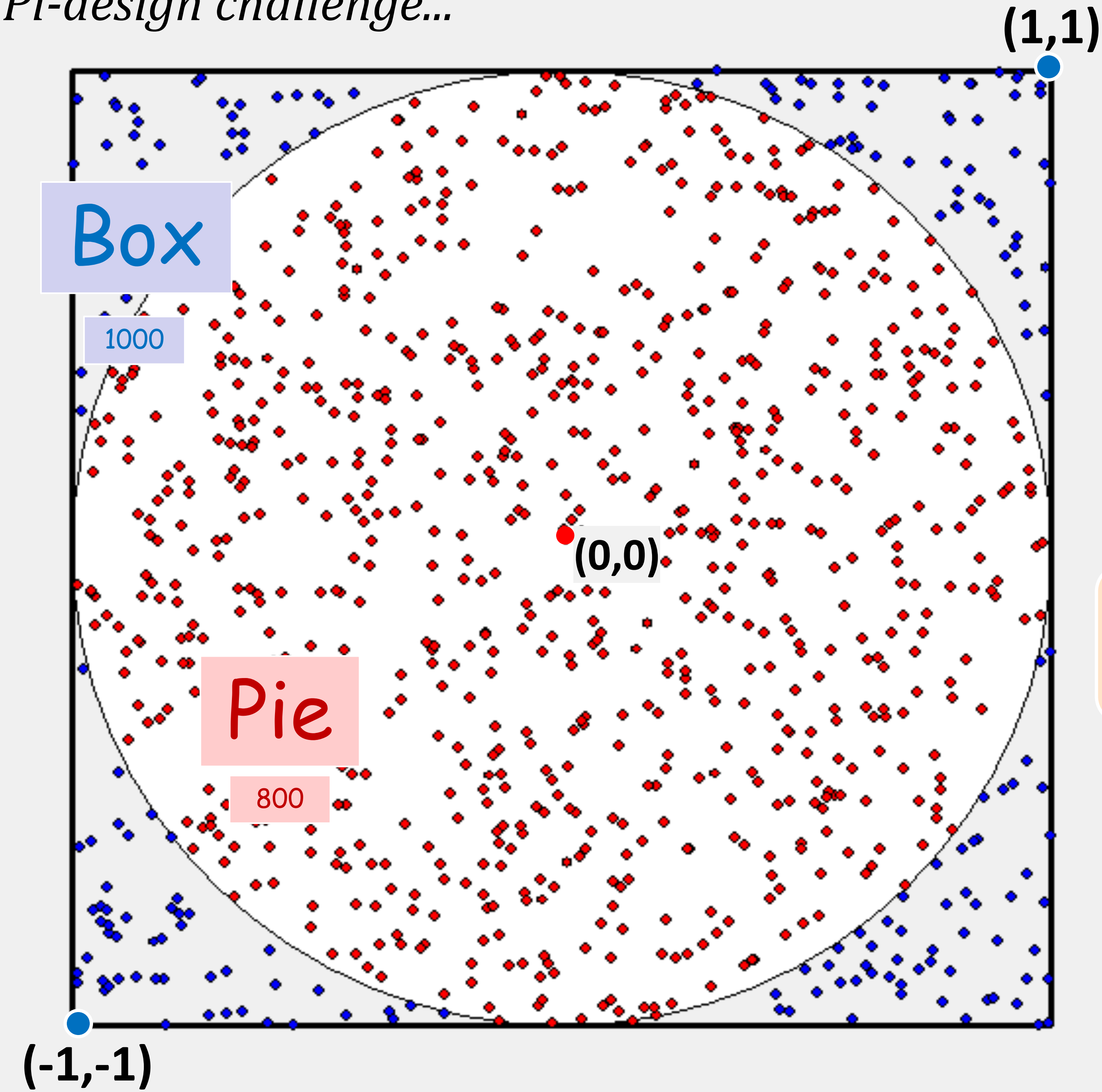


Estimating π from pie?

What if we just throw darts at this picture?

Name(s) ______________

Pi-design challenge...



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

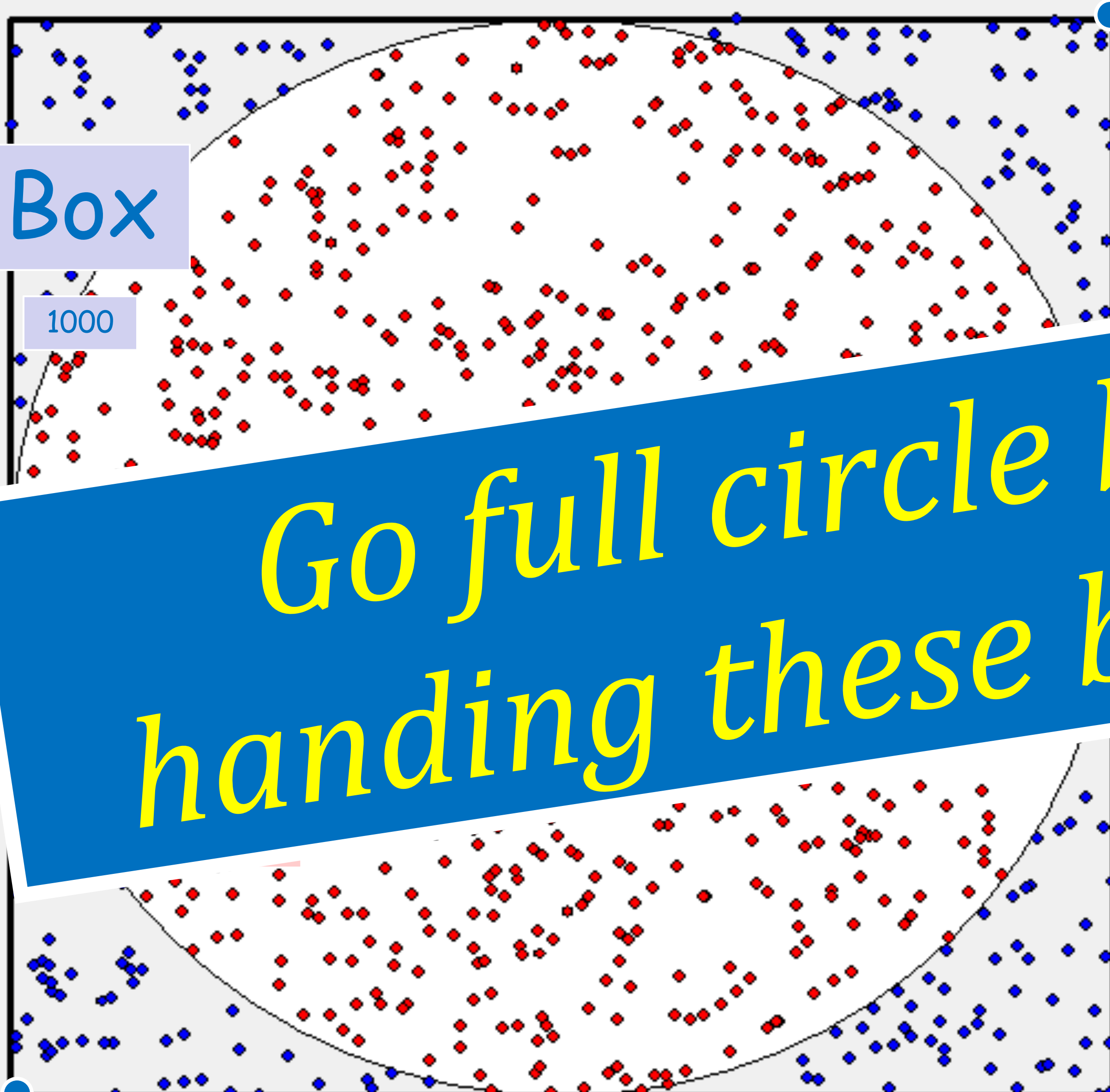
How big is a **side** of the square? its area?

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

How do these help!?!!

Name(s) ______________________

Pi-design challenge...



Estimating
π from pie?

(1) Suppos… w
…re.
…re.)

…n
…cle.

… & throws (1000)?

Go full circle by handing these back!

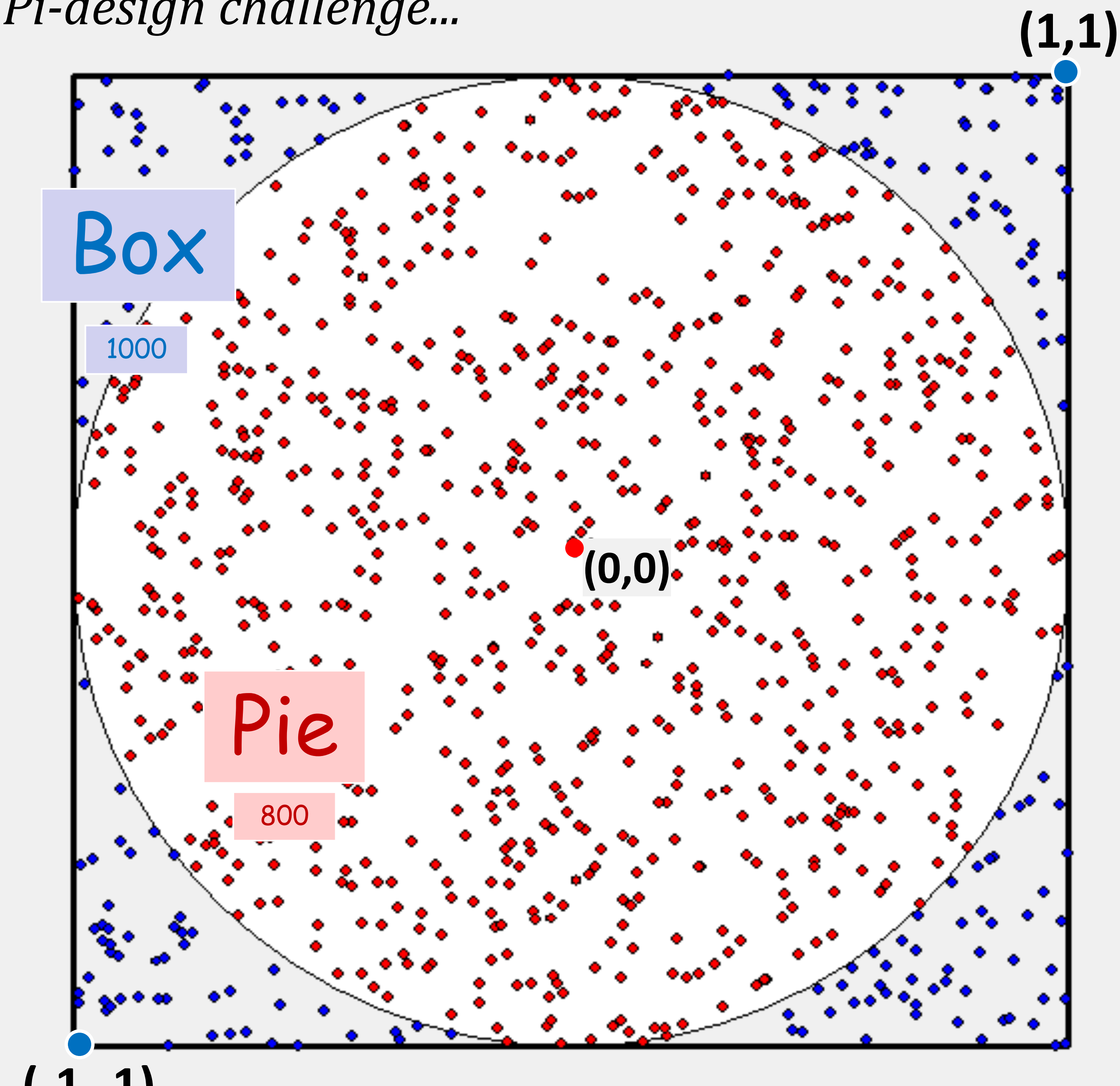
Hints

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?

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

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

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…

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

Better π with less work?

Leibniz's formula...

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

```python
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...

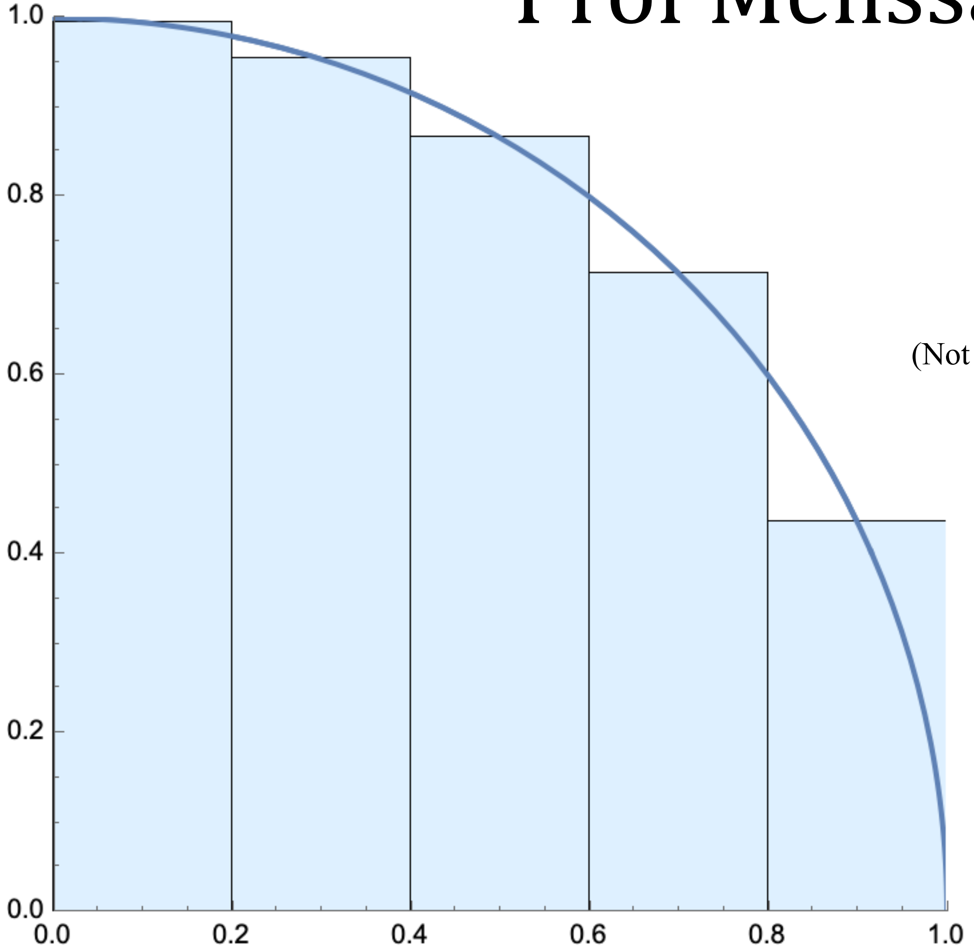
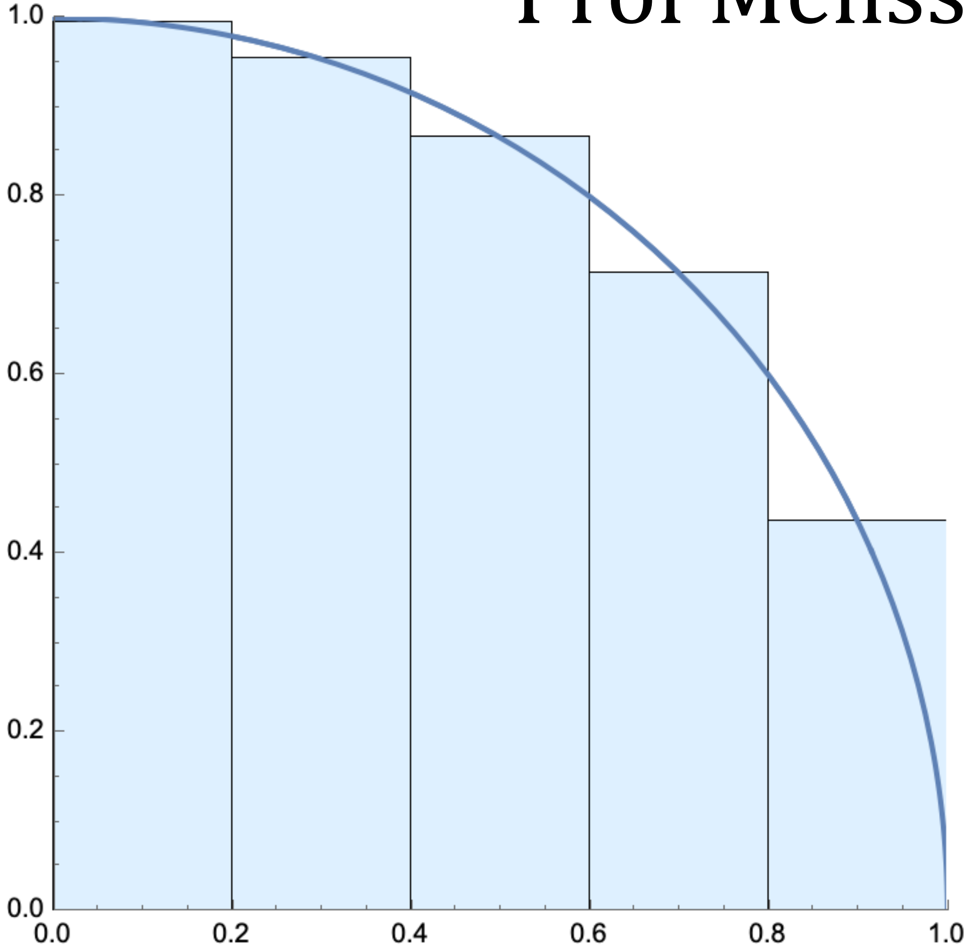


Figure out an easy way,
math genius not required!

(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...

```python
def slicepi(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

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

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

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

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

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

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

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

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

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)
```

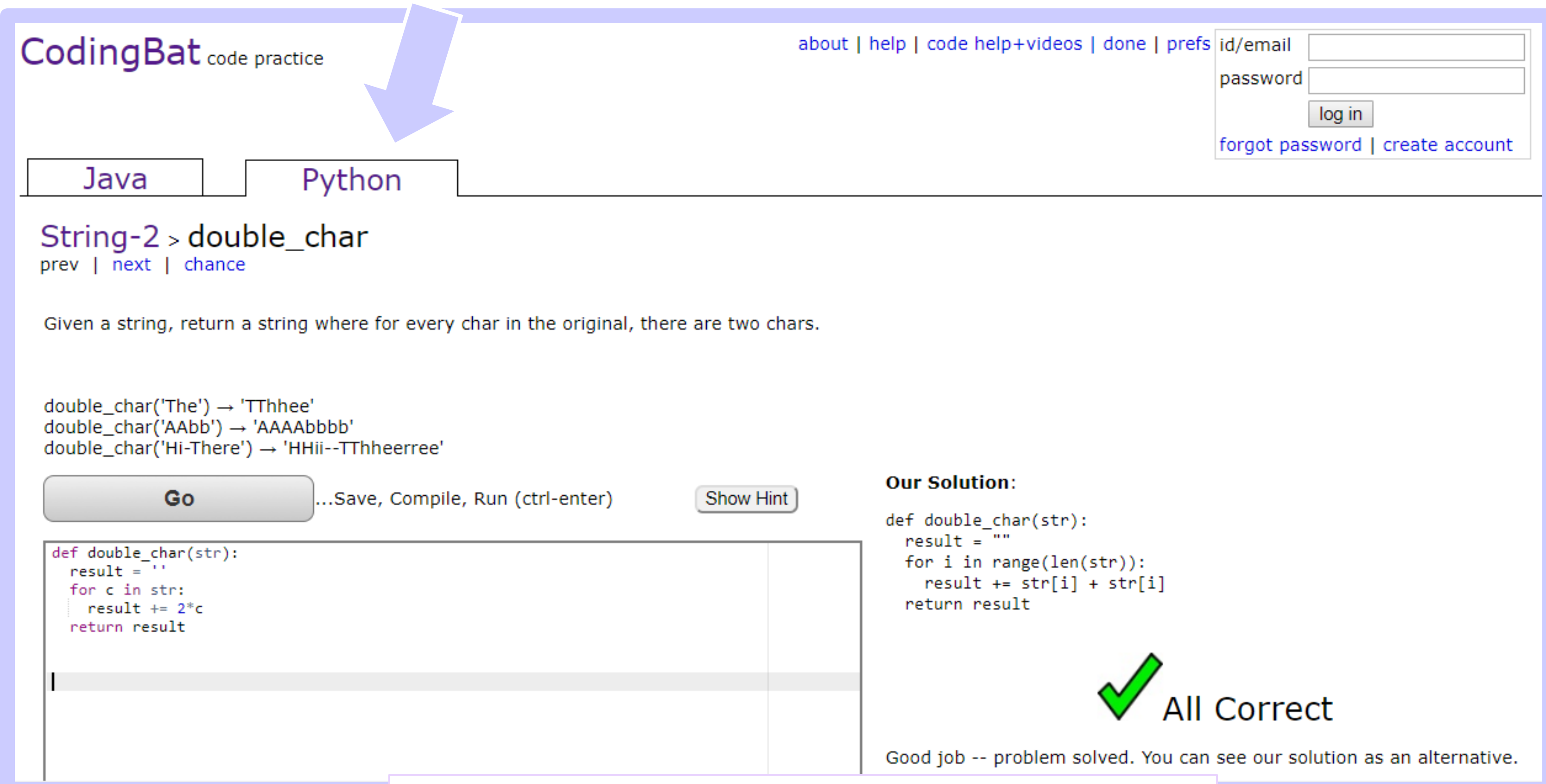
Homework 8 preview

#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...



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!

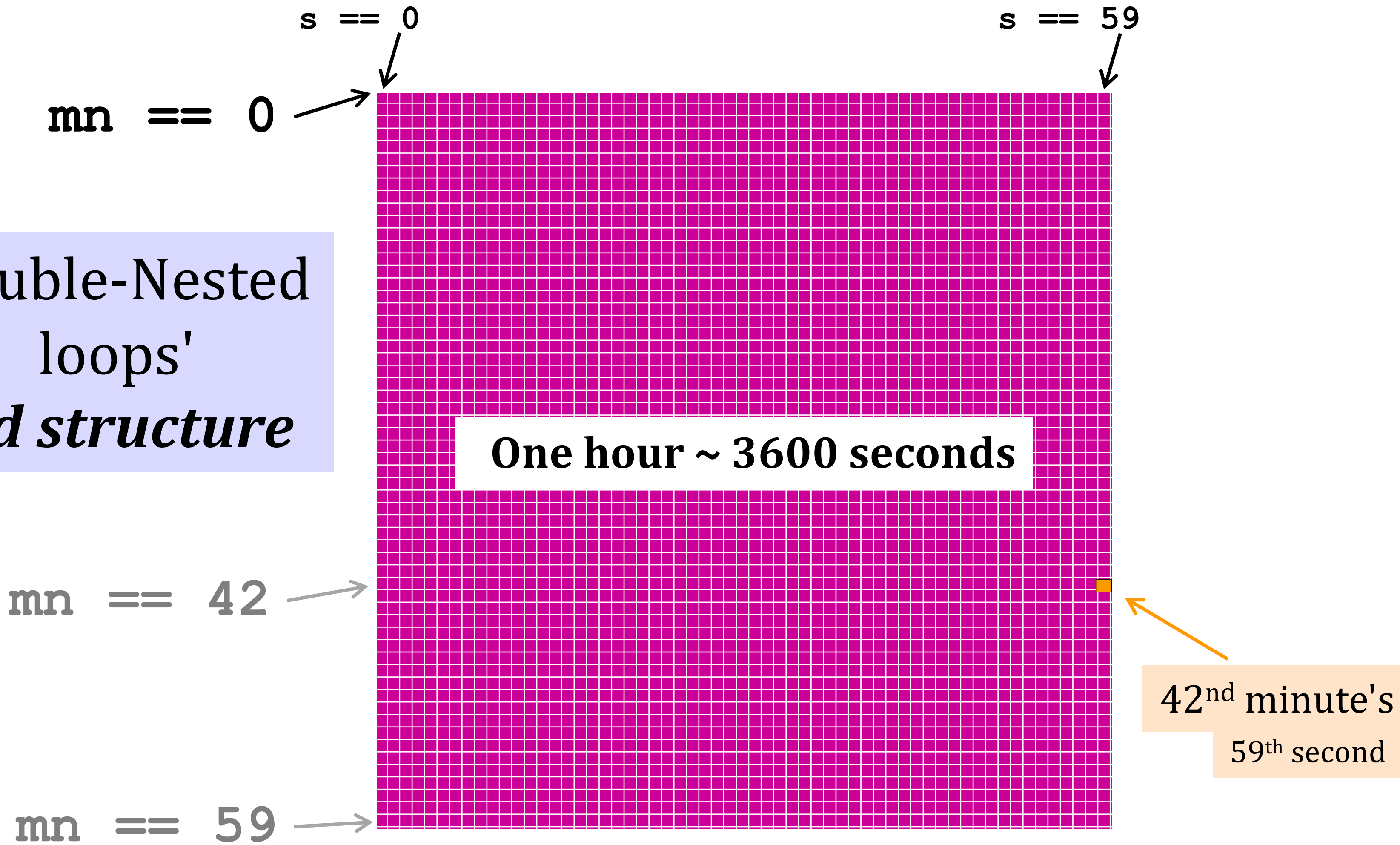
Nested loops

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()
```

Double-Nested loops'

2d structure




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

hour()

Creating 2D structure ~ in ASCII

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

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

Wait! this needs something more…

Creating 2D structure ~ in ASCII

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

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

list

list

Hmmm...

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

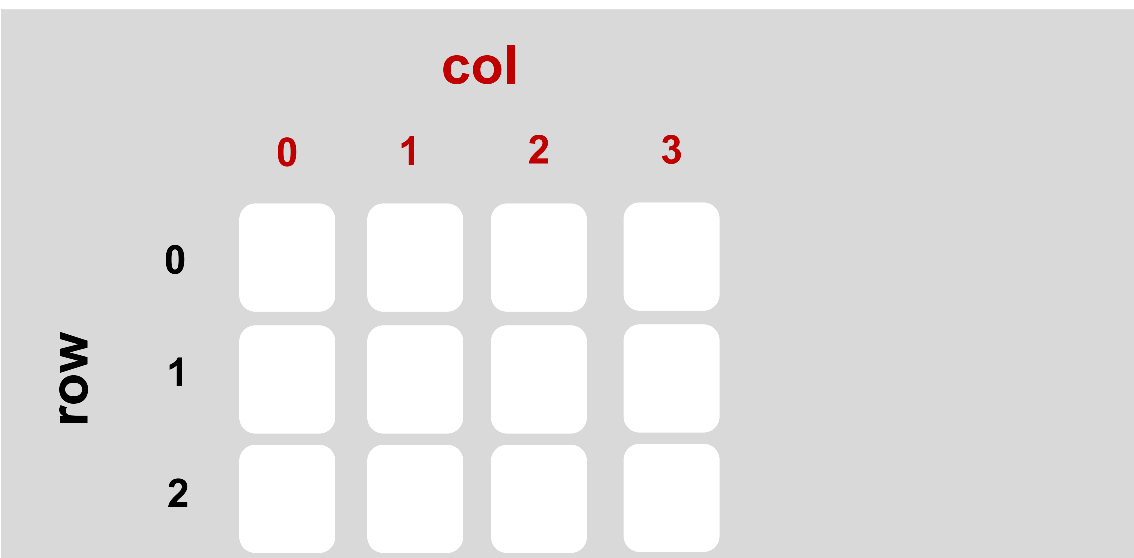
col

row	0	1	2	3
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()
```



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

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

B

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

C

```python
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			#		

```
if
```

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
```

Match!

What code creates the fourth, unmatched ASCII pattern?

D
```
if r+c < 5 :
```

answers...

A
```
for r in range(3):        [0,1,2]
  for c in range(6):      [0,1,2,3,4,5]
    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 — **C**

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

3 — **A**

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

4 — **B**

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`

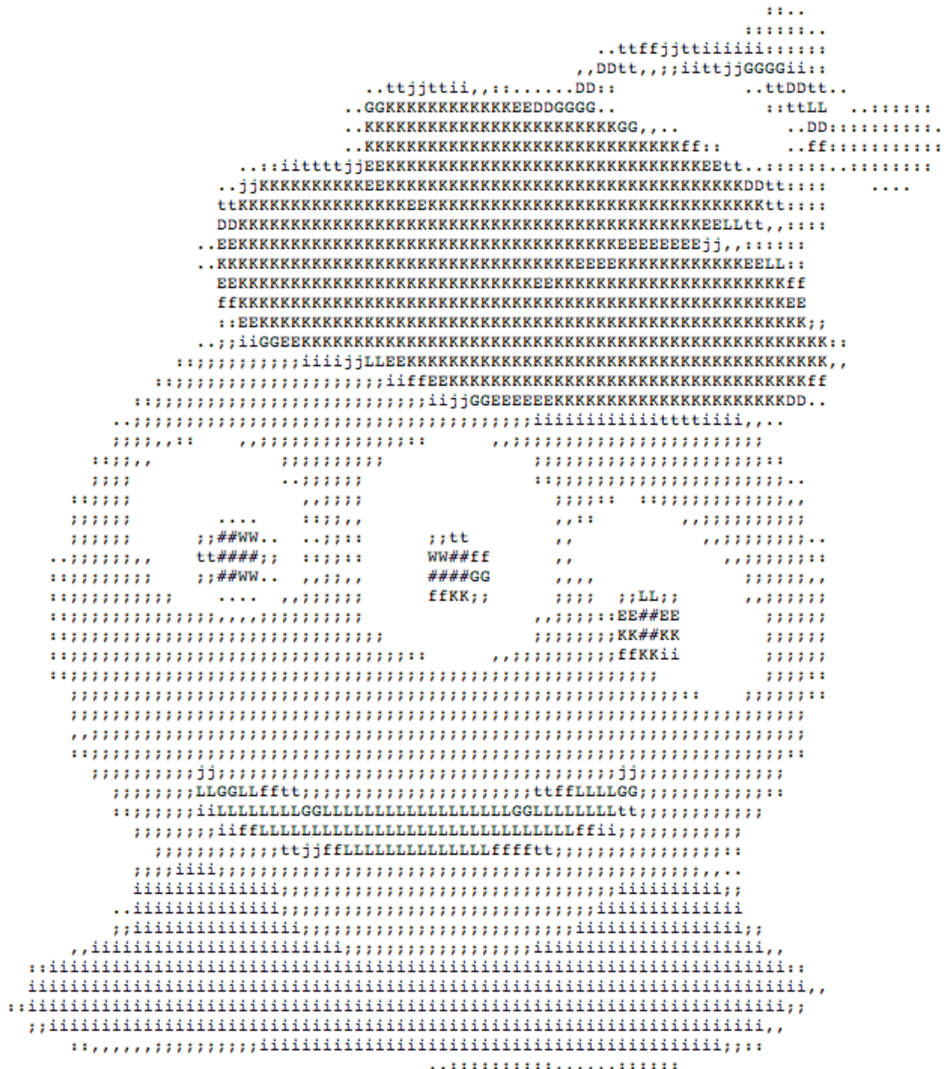
******

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!



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!

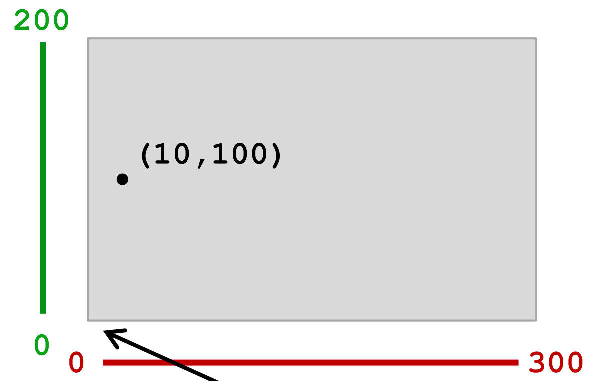
Python and images

```
from cs5png import *
```

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

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

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



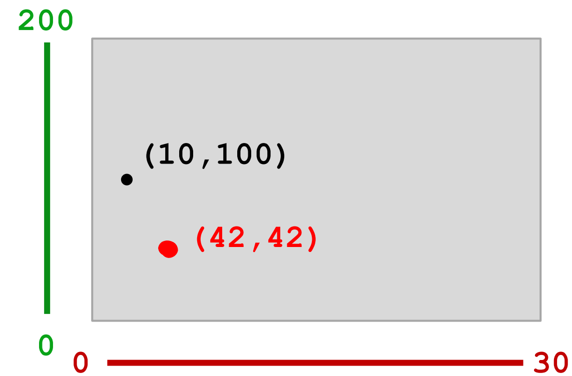
(0,0) is in the usual place!

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...

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?

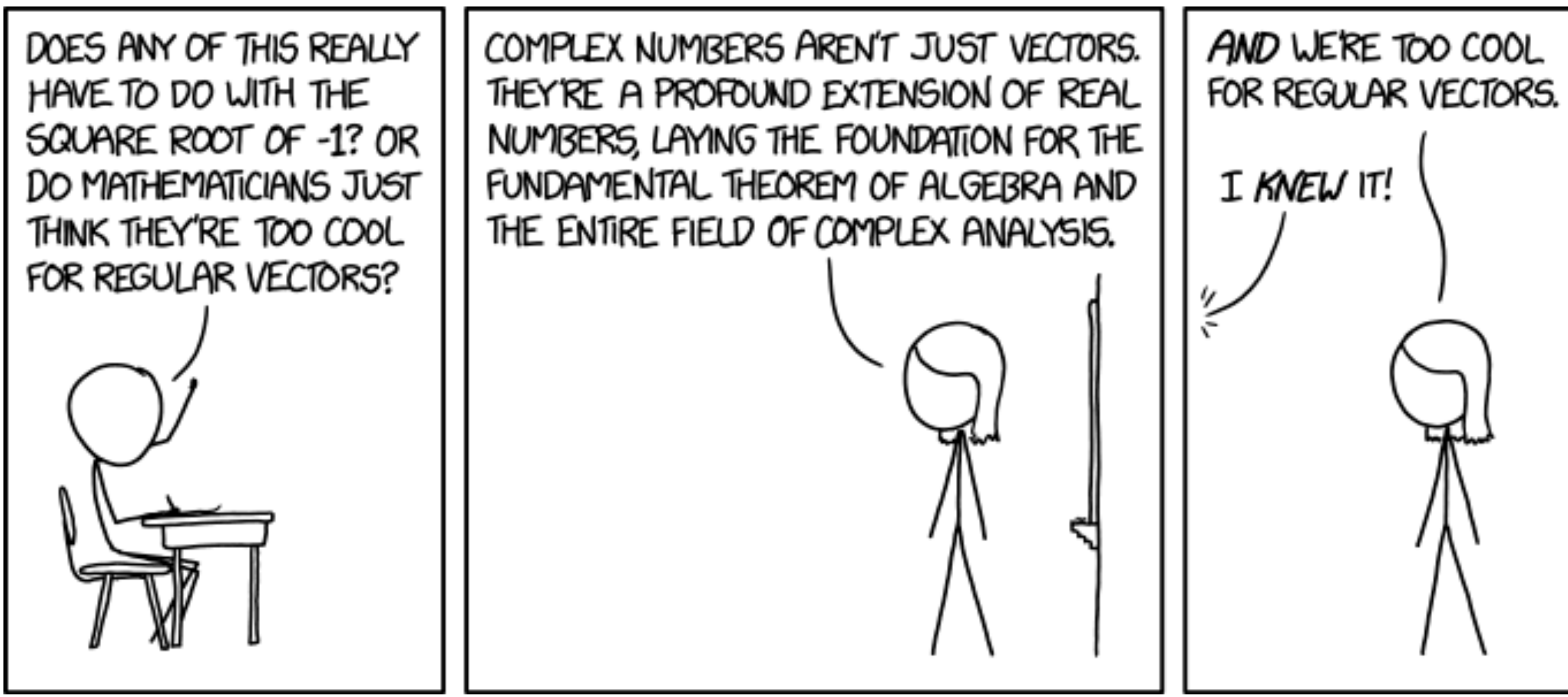
Complex Numbers!

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

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

i *can't believe this!*

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



xkcd no. 2028

Complex Numbers!

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

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

i *can't believe this!*

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

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

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

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

(3-4j)

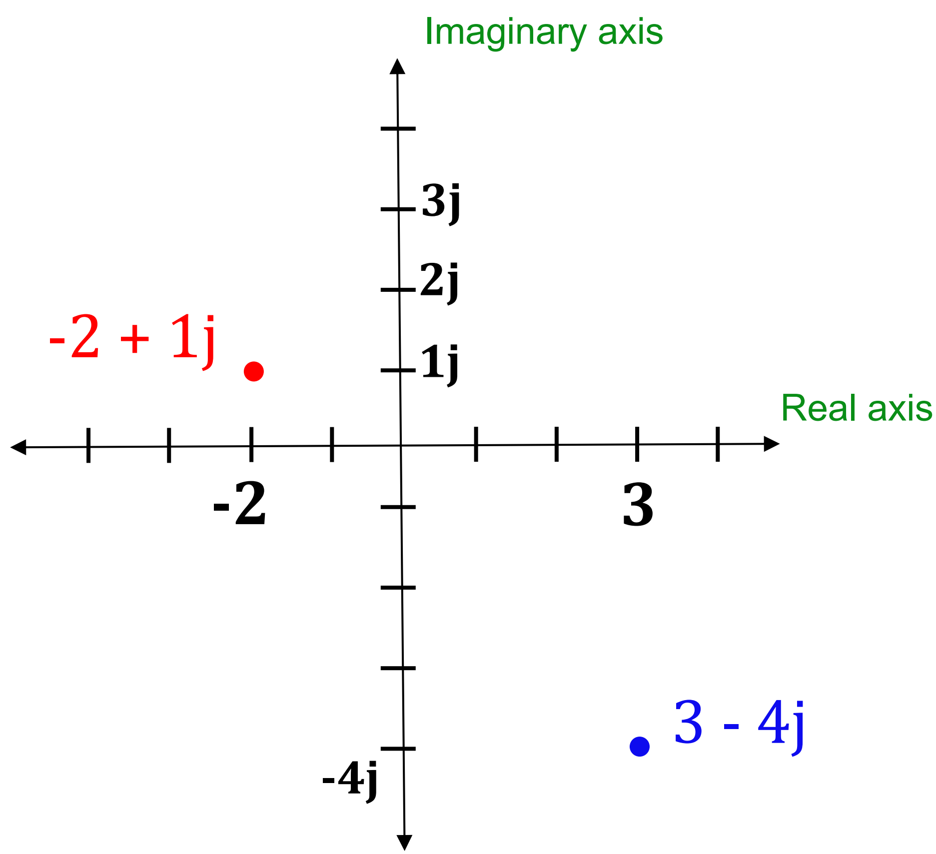
Complex Numbers!

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

i *can't believe this!*

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

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



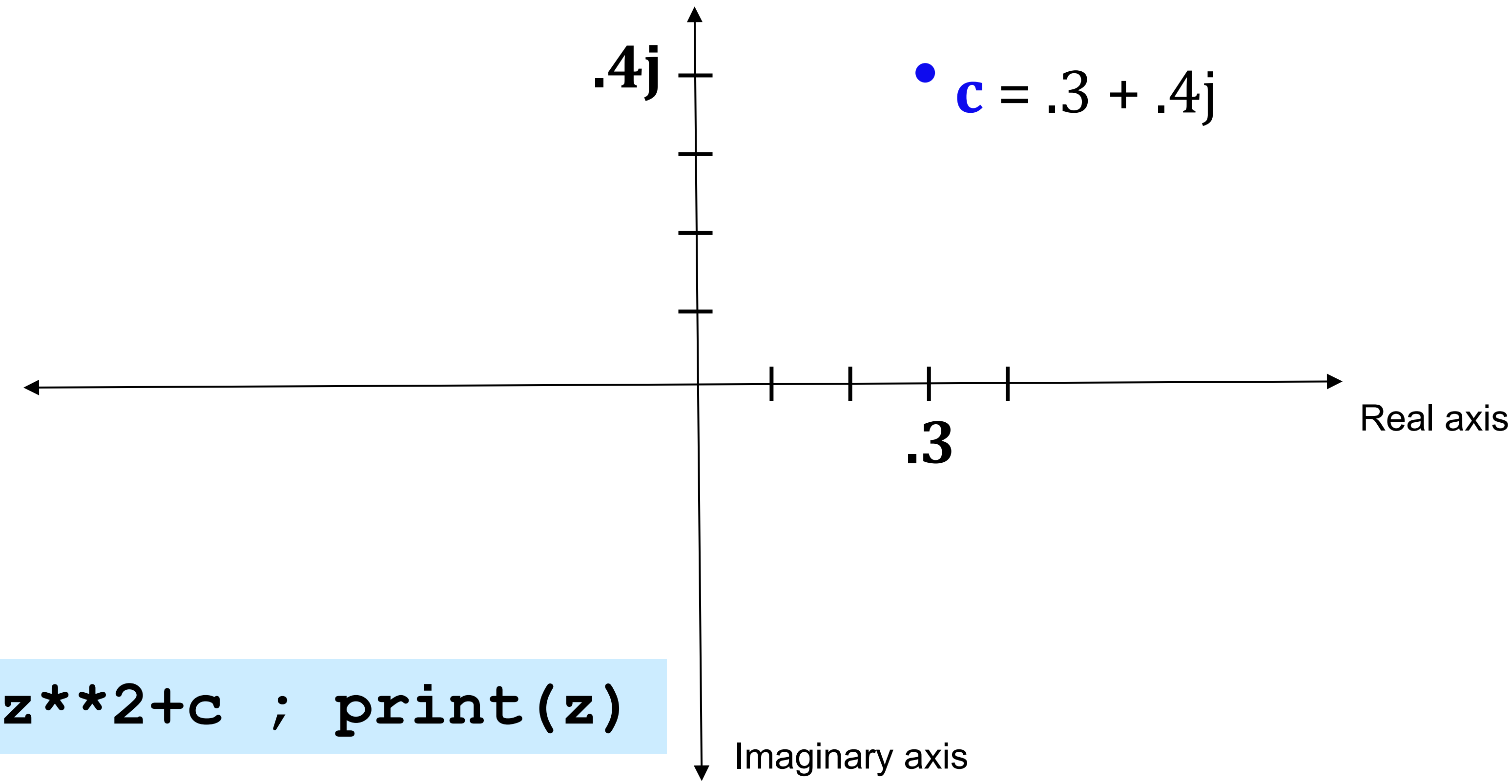
(c**2)**2 vs c = c**2; c**2

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

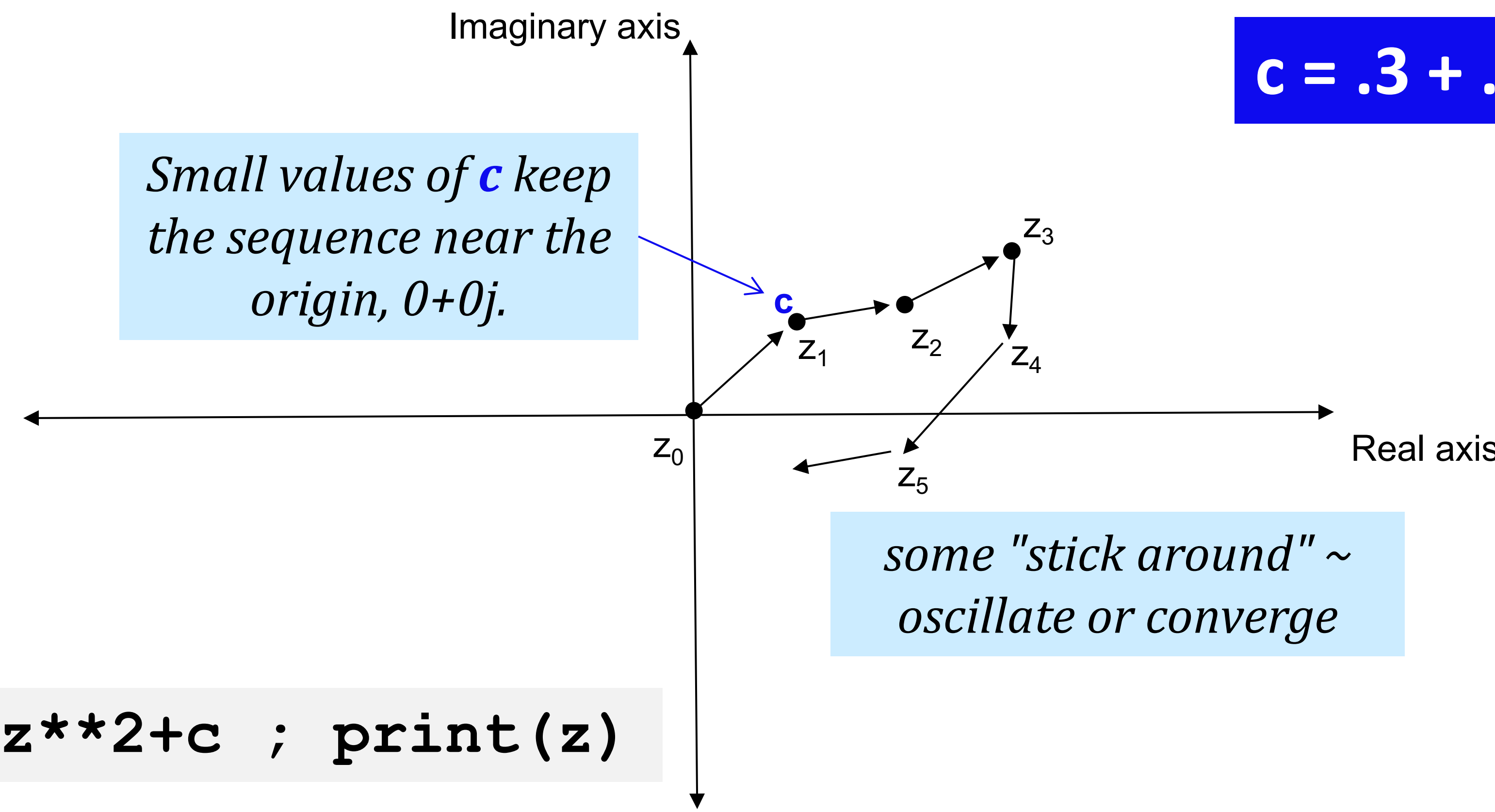
Consider an ***update rule***
for all complex numbers ***c***

$$z_0 = 0$$

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

c = .3 + .4j

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



some "stick around" ~ oscillate or converge

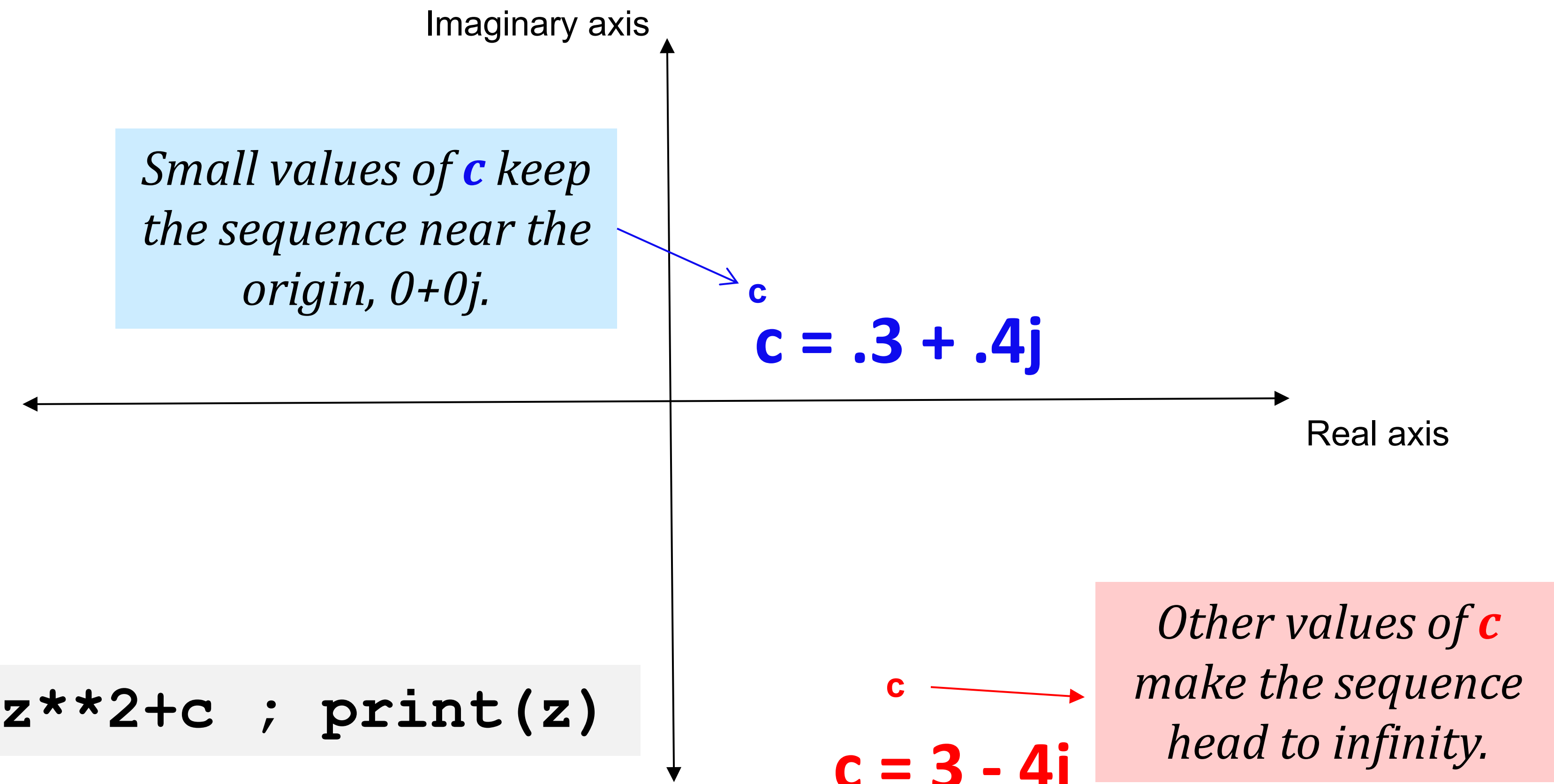
```
z = z**2+c ; print(z)
```

Mandelbrot Definition

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

$$z_0 = 0$$

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



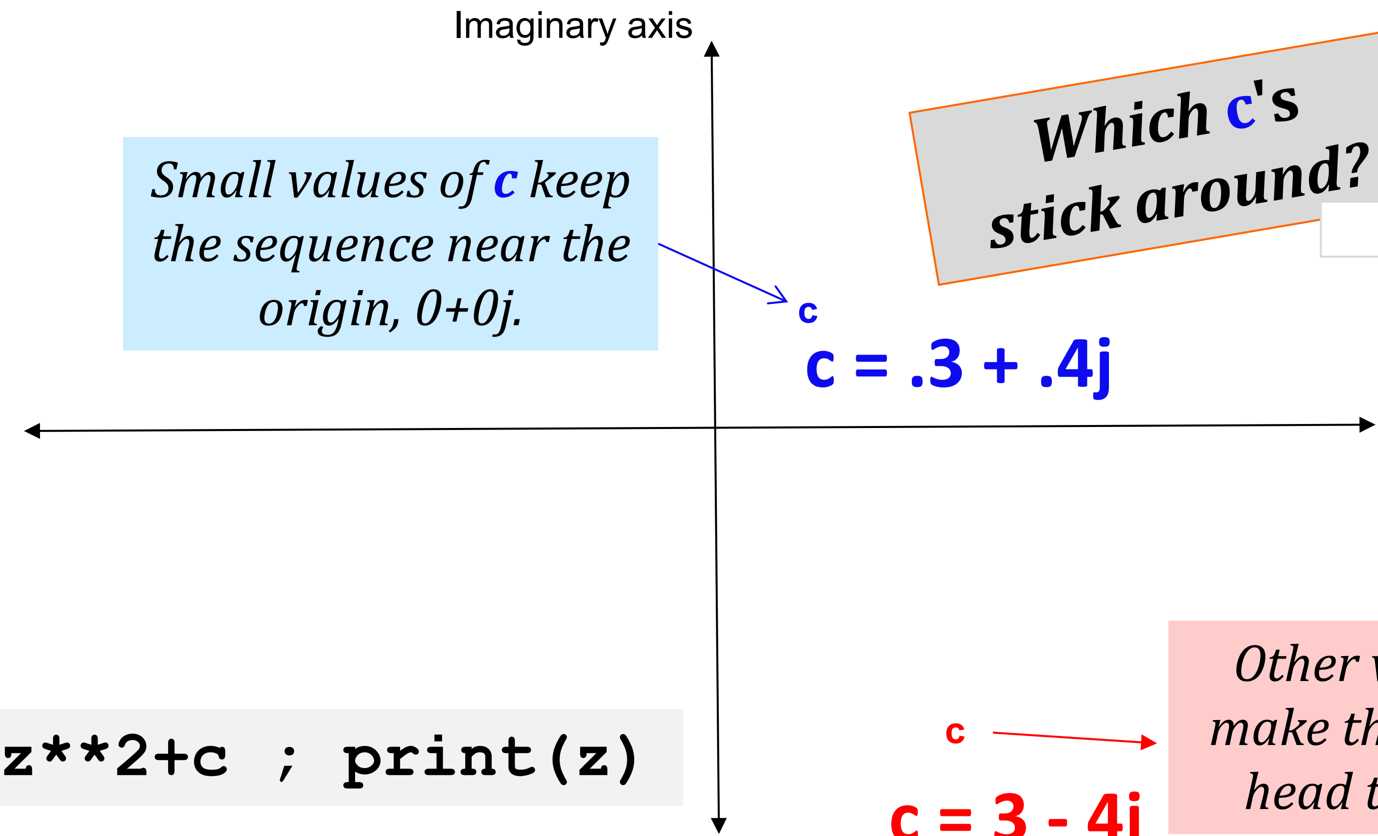

```
z = z**2+c ; print(z)
```

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?

- Mandelbrot

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

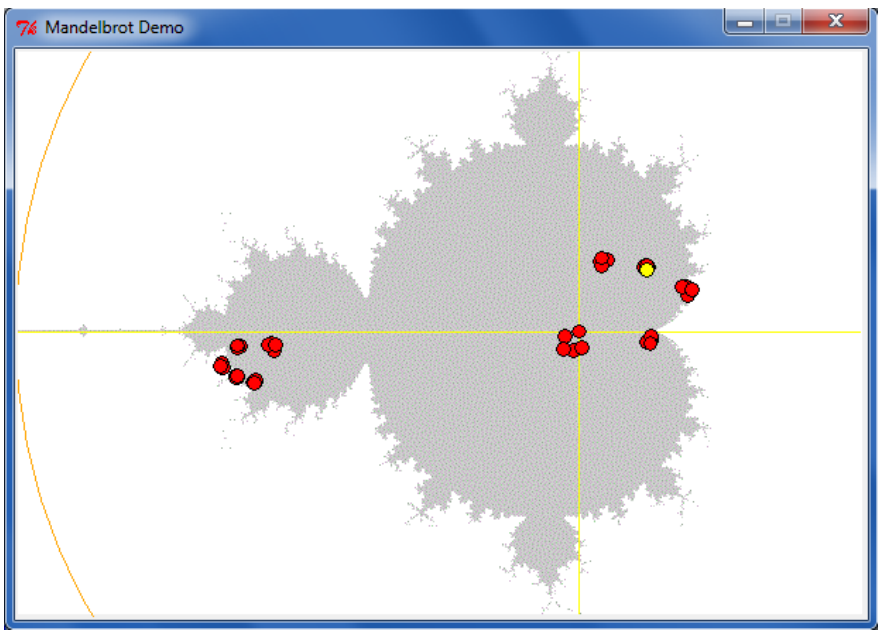
```
z = z**2+c ; print(z)
```

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 + -0.0546346462374 * 1j
iter # 4 : z = 0.256047527535 + -0.0303026920702 * 1j
iter # 5 : z = -1.14904739926 + -0.1784[illegible]6935 * 1j
iter # 6 : z = 0.0747849173552 [illegible]7964 * 1j
iter # 7 : z = -1.2691[illegible]5977 * 1j
iter # 8 : [illegible]88 * 1j
iter # 9 [illegible]08 * 1j
iter # 10 [illegible]56431 * 1j
iter # 11 [illegible]602 * 1j
iter # 12 [illegible]14 * 1j
iter # 13 [illegible]5207 * 1j
iter # 14 : [illegible]227677689 * 1j
iter # 15 : [illegible] + -0.180471770237 * 1j
iter # 16 : [illegible]5360076381 + 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
```

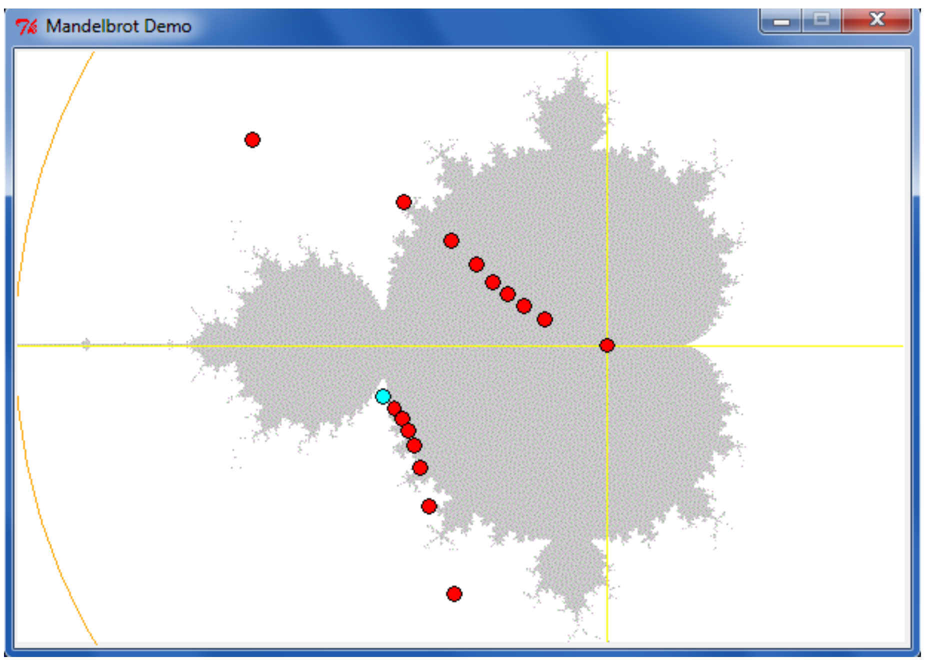
some ***c***'s stick around

Lab 8: *the Mandelbrot Set*

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

$$z_0 = 0$$

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

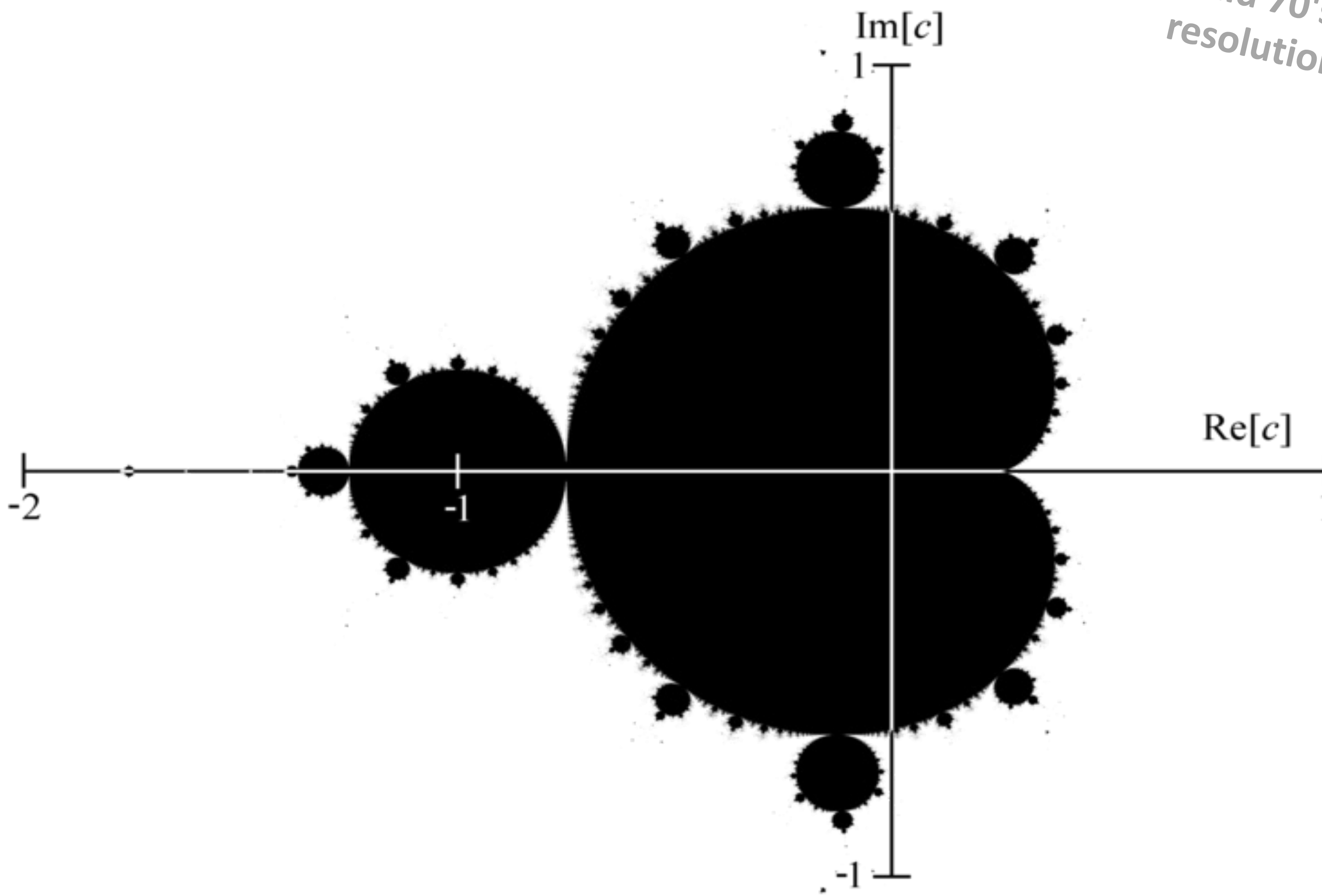



```
Click to choose c.
c is -0.757929883139 + -0.172932330827 * 1j

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 + 0.131    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 :                     .543792492805 * 1j
iter # 14 : z              6018 + 0.48336747636 * 1j
iter # 15 : z   .316174815348 + -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

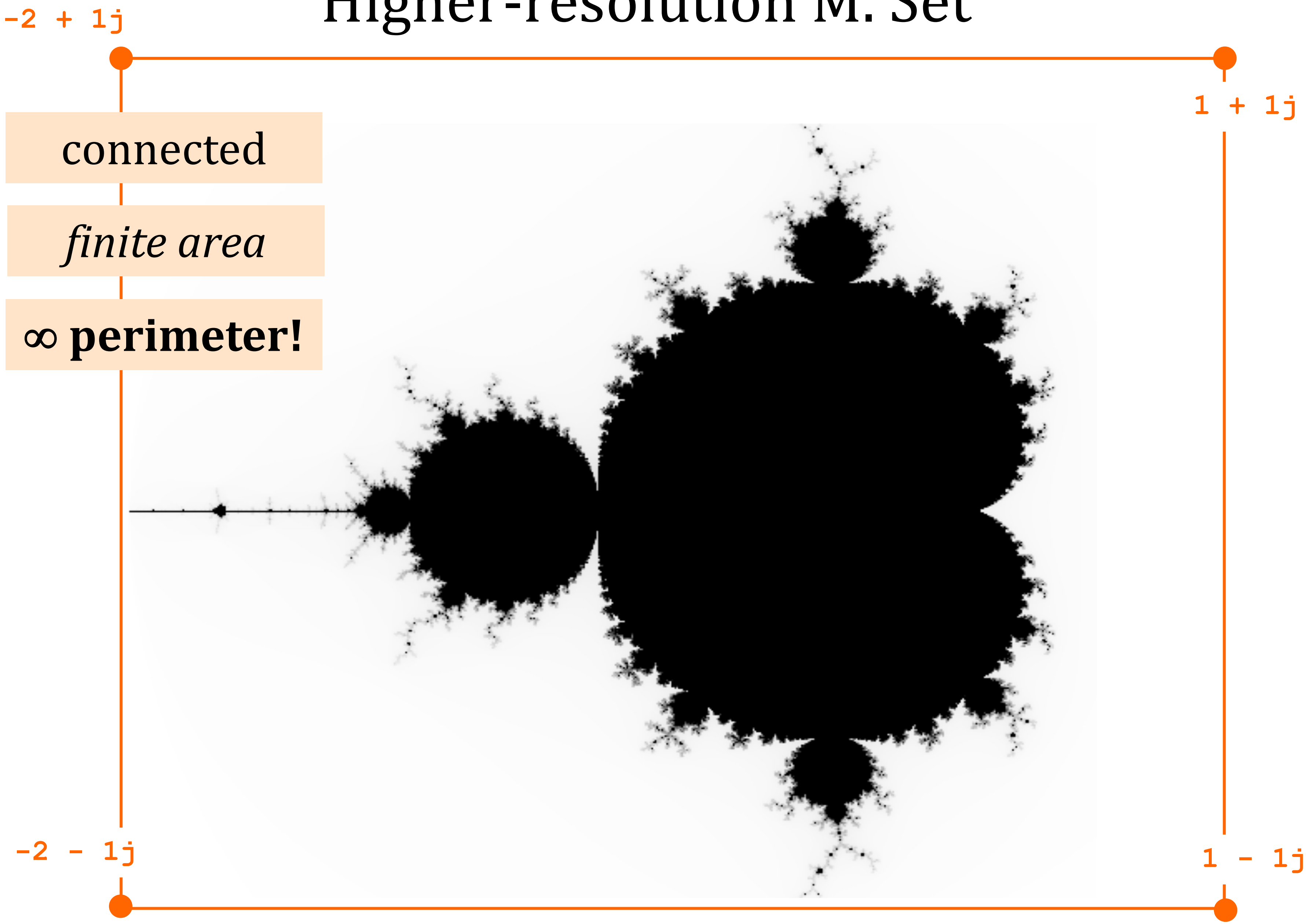
The shaded area are points that do ***not*** diverge for `z = z**2 + c`

Higher-resolution M. Set

connected

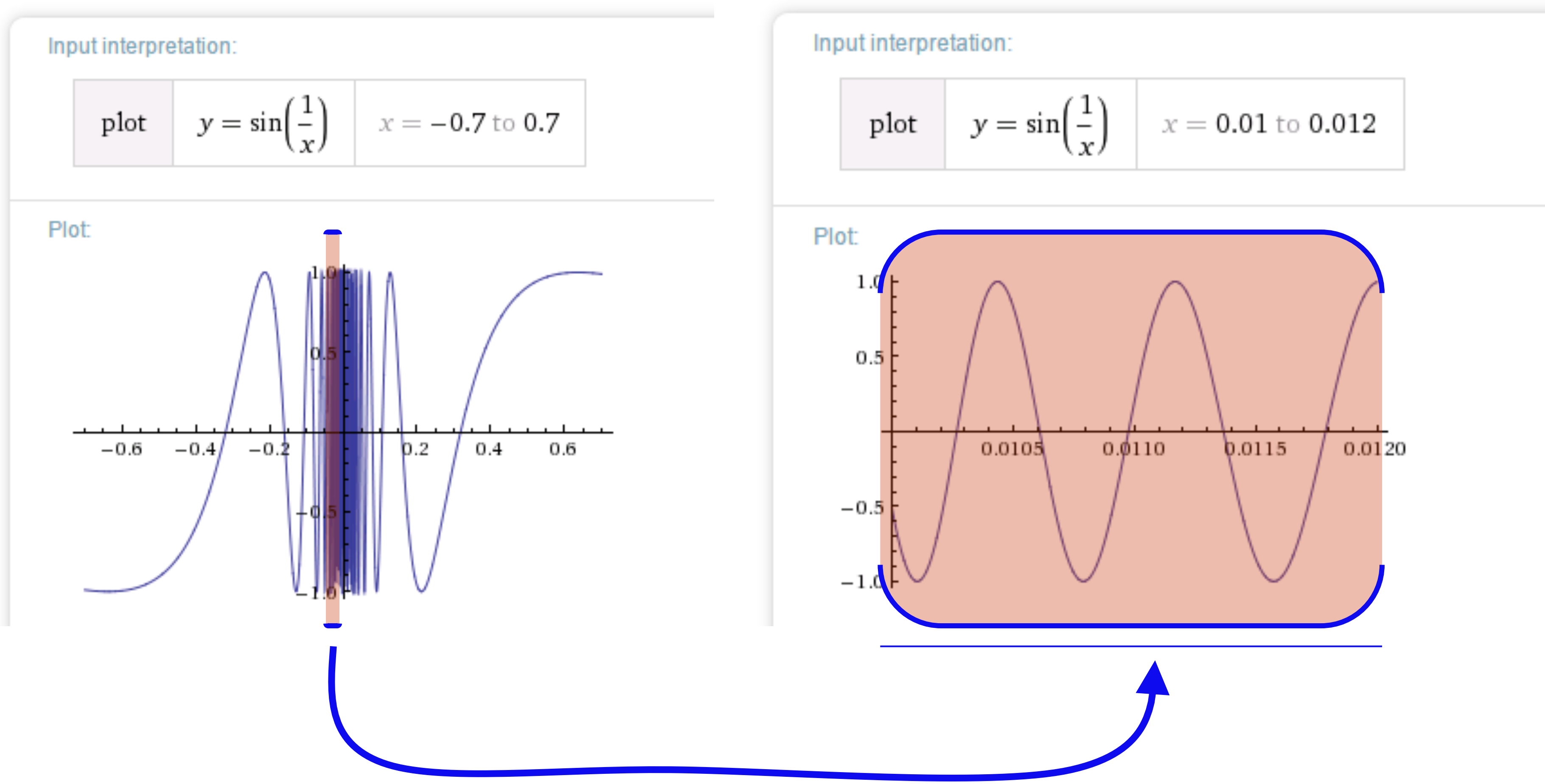
finite area

∞ perimeter!



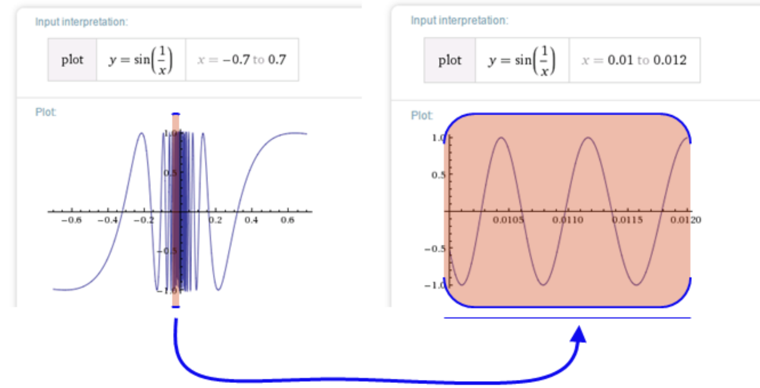
The black pixels are points that do ***not*** diverge for **`z = z**2 + c`**

Chaos?



Complex things always consisted of simple parts…

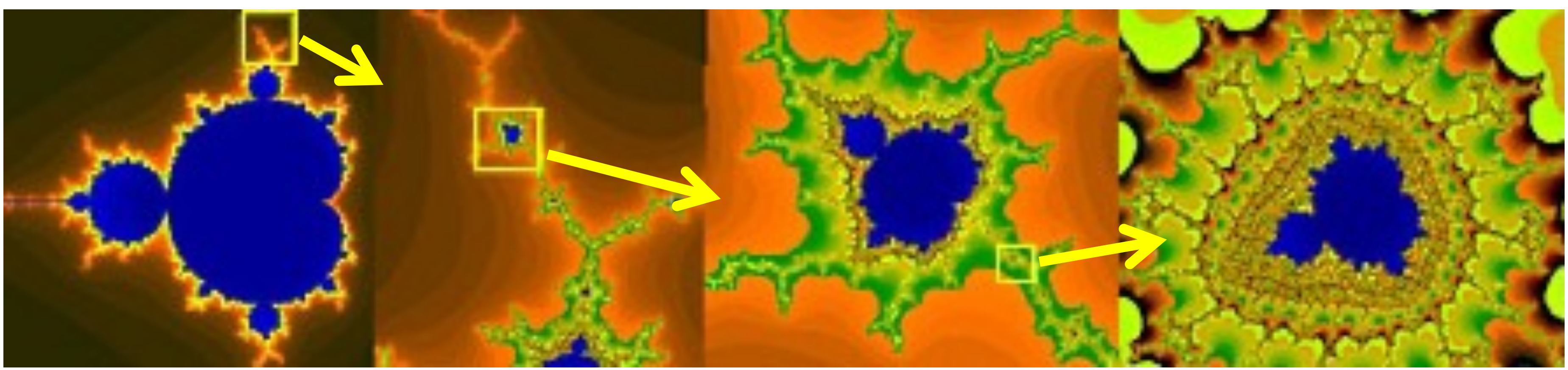
Before the M. Set, complex things
were made of simple parts:



Chaos!

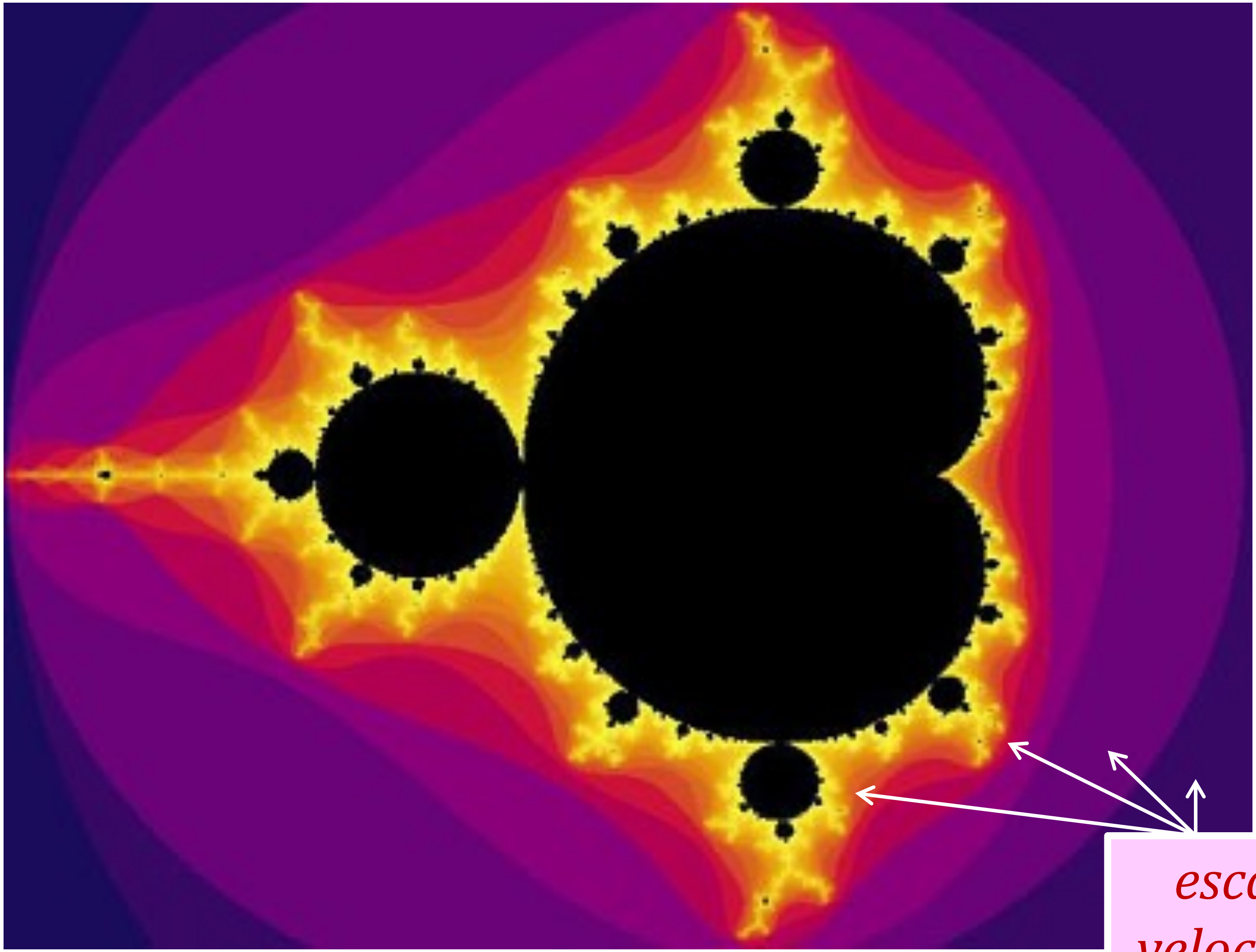
This was a *"naturally occurring"* object where
zooming uncovers ***more*** detail, not less:

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



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

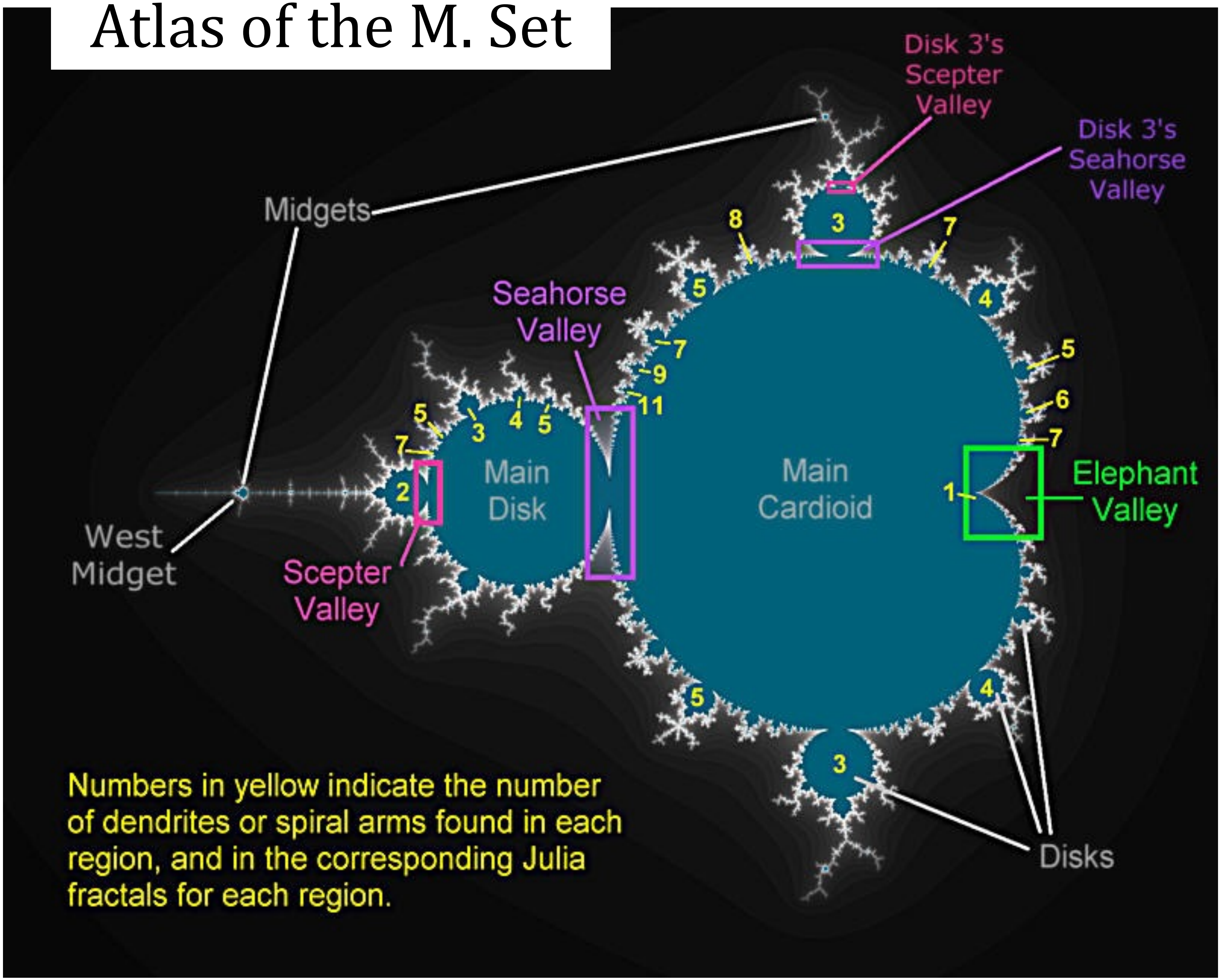
The M. Set pixels are points that do ***not*** diverge for `z = z**2 + c`



What are these colors?

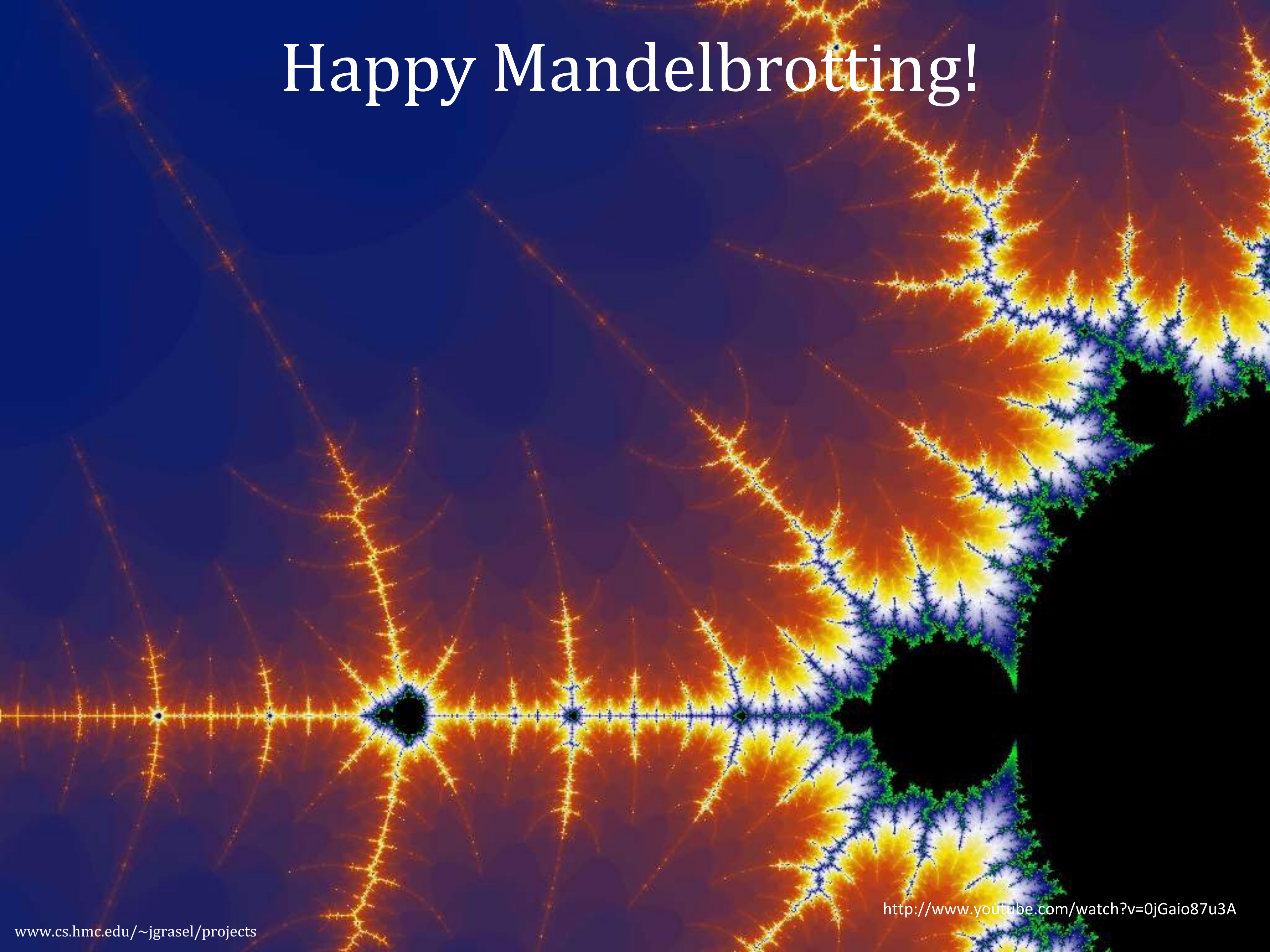
??

Atlas of the M. Set



In the *Seahorse Valley*….

Happy Mandelbrotting!



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

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