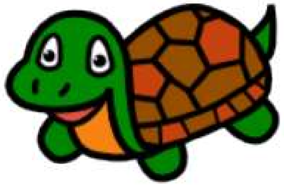
Cyriak: *conceptually disruptive* recursion...

Fractals and Turtles

CS 5
alien on
strike!

CS 5 green mascot
representing today's
terrestrial theme



CS 5 Today

hw2 due Mon. 9/23

Lots of tutoring...

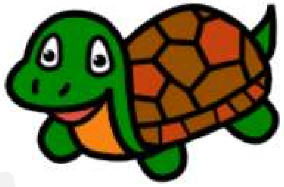
How random!

Cyriak: *conceptually disruptive* recursion...

Fractals and Turtles

CS 5
alien on
strike!

CS 5 green mascot
representing today's
terrestrial theme



Applications!

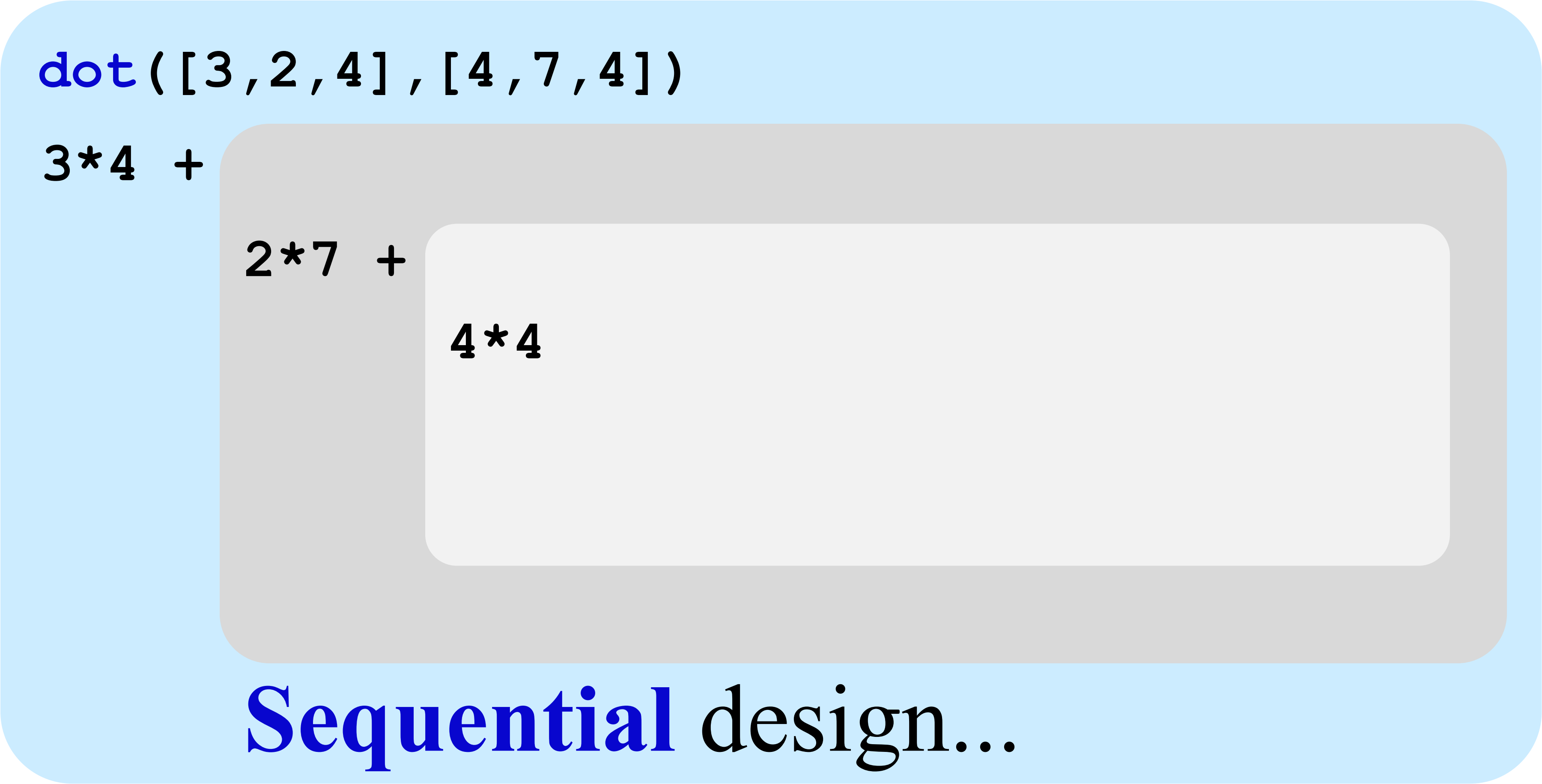
CS 5

hw2 due Mon. 9/23

Lots of tutoring...

How random!

```
dot([3,2,4],[4,7,4])
```



```
dot([3,2,4],[4,7,4])
3*4 +
      2*7 +
            4*4
```

Sequential design...

```
dot([3,2,4],[4,7,4])
```

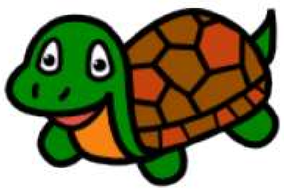
```
dot([3,2,4],[4,7,4])

3*4 + dot([2,4],[7,4])
```

Recursive design...

dot ...

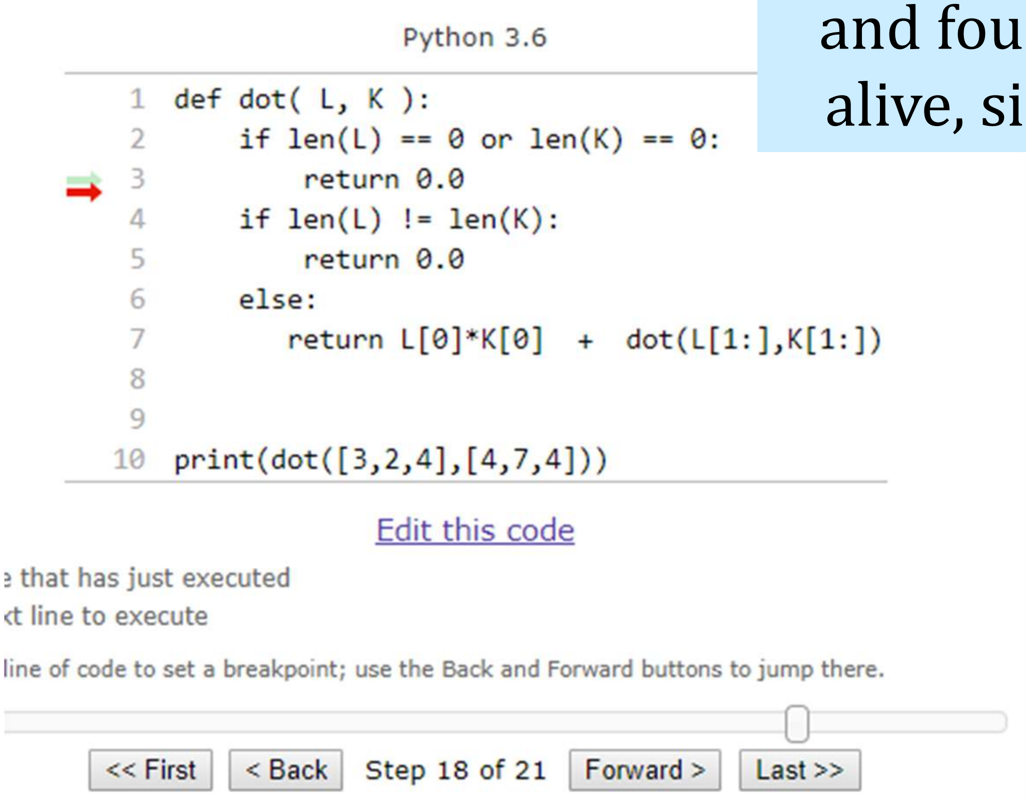
```
def dot( L, K ):
    if len(L) == 0 or len(K) == 0:
        return 0.0
    if len(L) != len(K):
        return 0.0
    else:
       return L[0]*K[0]  +  dot(L[1:],K[1:])
```

```
dot([3,2,4],[4,7,4])                                L = [3,2,4] and K = [4,7,4]
3*4 + dot([2,4],[7,4])                              L = [2,4] and K = [7,4]
      2*7 + dot([4],[4])                            L = [4] and K = [4]
            4*4 + dot([],[])                        L = [ ] and K = [ ]
                  0.0
            16.0
      30.0
42.0
```



pythontutor.com

There are four different values of L and four different values of K – all alive, simultaneously, in the stack



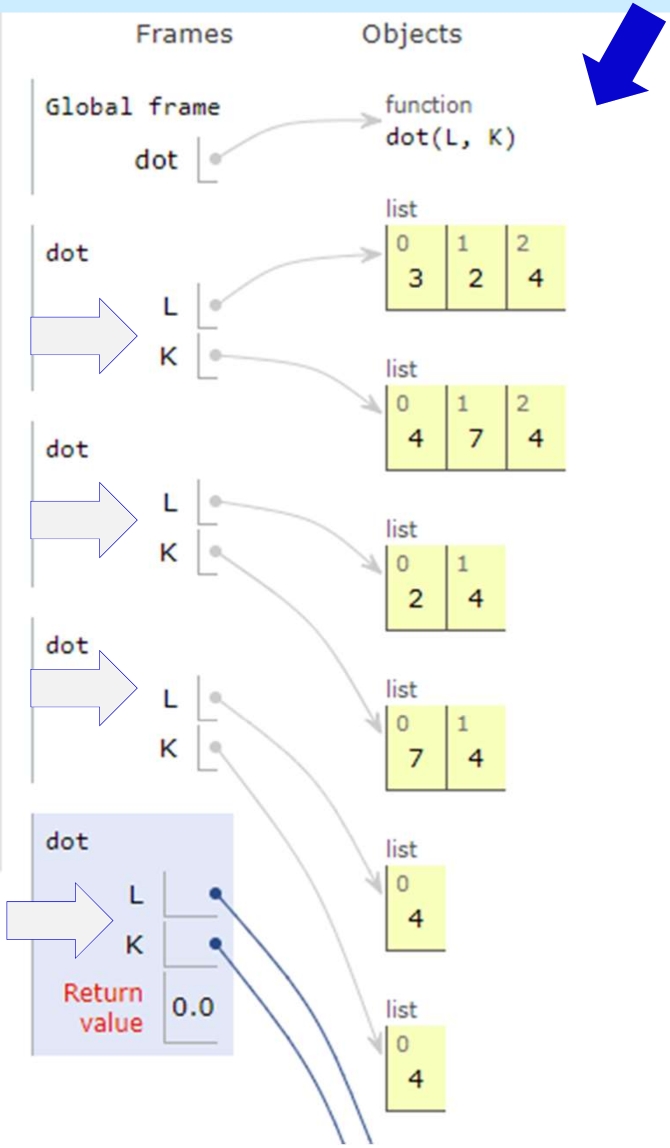
Python 3.6

```
def dot( L, K ):
    if len(L) == 0 or len(K) == 0:
        return 0.0
    if len(L) != len(K):
        return 0.0
    else:
       return L[0]*K[0]  +  dot(L[1:],K[1:])


print(dot([3,2,4],[4,7,4]))
```

e that has just executed

xt line to execute

line of code to set a breakpoint; use the Back and Forward buttons to jump there.



Seeing the "stack" ...

Recursion's idea:

You handle the FIRST

Recursion handles the REST

Recursion's idea:

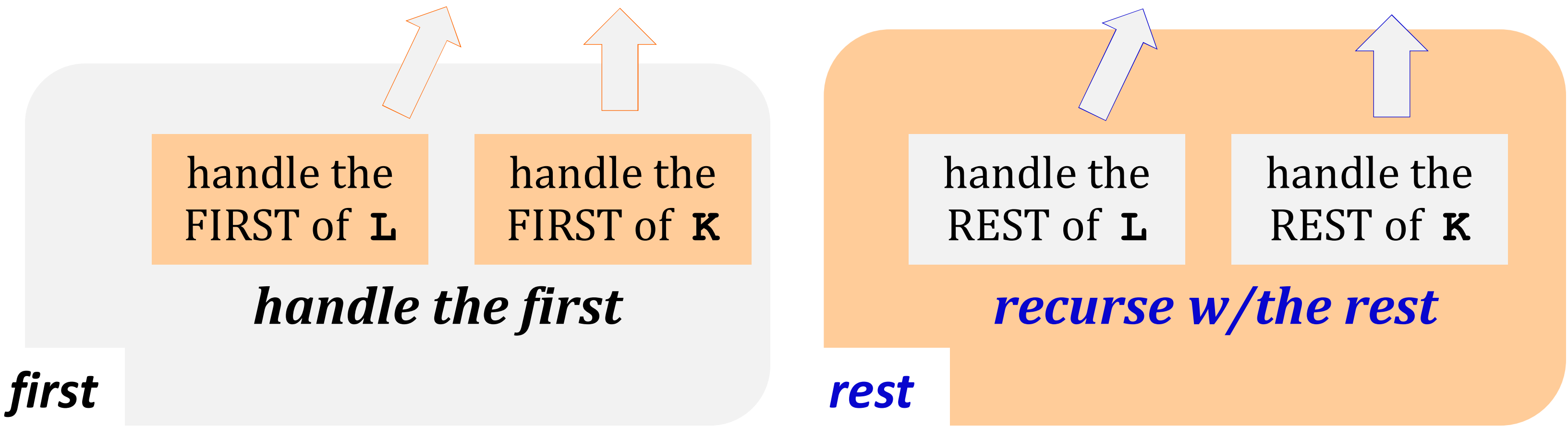
You handle the FIRST

Recursion handles the REST

```
def dot( L, K ):
```

combine

```
        return L[0]*K[0]  +  dot(L[1:],K[1:])
```



handle the FIRST of `L`

handle the FIRST of `K`

handle the first

first

handle the REST of `L`

handle the REST of `K`

recurse w/the rest

rest

Recursion's idea:

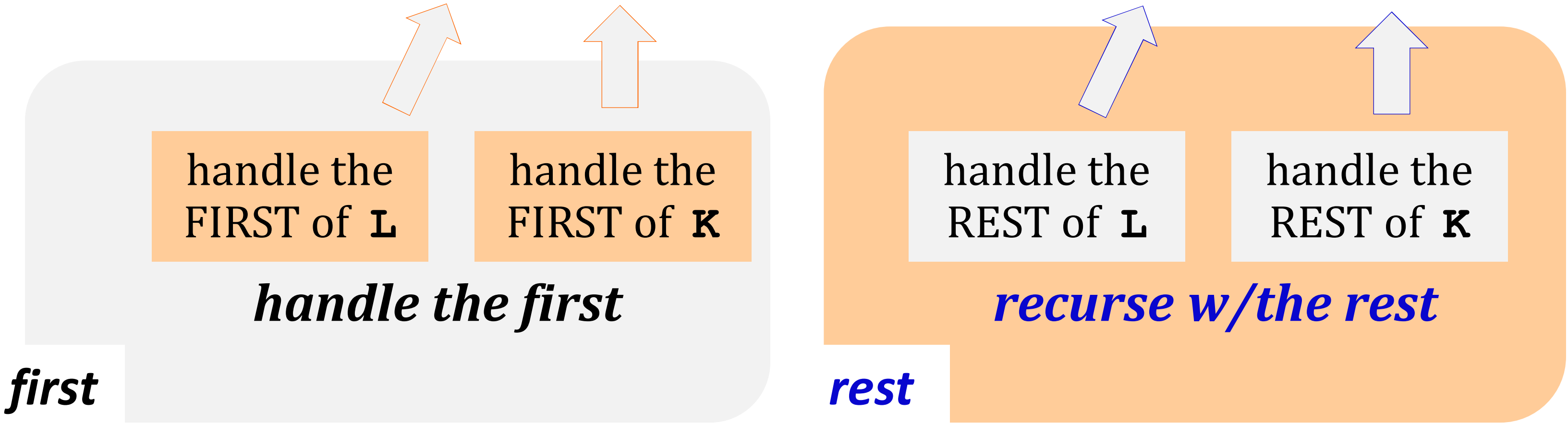
You handle the FIRST

Recursion handles the REST

```
def dot( L, K ):
    if len(L) == 0 or len(K) == 0:
        return 0.0

    if len(L) != len(K):
        return 0.0

    else:
        return L[0]*K[0]  +  dot(L[1:],K[1:])
```

Base Cases

combine



handle the FIRST of **L**

handle the FIRST of **K**

handle the first

first

handle the REST of **L**

handle the REST of **K**

recurse w/the rest

rest

Some *random* asides…

`import random`	allows use of **dir(random)** and **help(random)**
`from random import *`	all random functions are now available!

Some *random* asides...

```
import random
from random import *
```

allows use of **dir(random)** and **help(random)**

all random functions are now available!

```
choice( L )
```

chooses 1 element from the sequence L

```
choice('mudd')
```

... or 1 character from a string

```
choice(['cmc','scripps','pitzer','pomona'])
```

Some *random* asides...

```
import random
```

allows use of **dir(random)** and **help(random)**

```
from random import *
```

all random functions are now available!

```
choice( L )
```

chooses 1 element from the sequence L

```
choice('mudd')
```

... or 1 character from a string

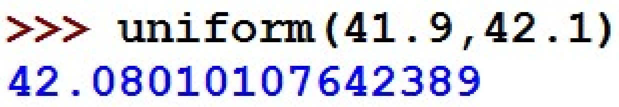
```
choice(['cmc','scripps','pitzer','pomona'])
```

```
list(range(5))    →  [0,1,2,3,4]
list(range(1,5))  →  [1,2,3,4]
```

How would you get a random integer from 0 to 99 inclusive?

```
uniform(low,hi)
```

chooses a random **float** from low to hi



```
>>> uniform(41.9,42.1)
42.08010107642389
```

floats have **16** places of precision

Aargh – so close!

A *"random"* function…

```
from random import *

def guess( hidden ):
    """ tries to guess our "hidden" #
    """
    compguess = choice( list(range(100)) )

    if compguess == hidden:
        print('I got it!')

    else:
        guess( hidden )
```

Remember, this is [0,1,…,98,99]

print the guesses ?

slow down…

return the number of guesses ?

investigate **expected** # of guesses?!??

Recursive guess-counting

code available in hw2pr2

```python
from random import *
import time

def guess( hidden ):
    """ guessing game """
    compguess = choice( list(range(100)) )

    # print('I choose', compguess)
    # time.sleep(0.05)

    if compguess == hidden:   # at last!
        # print('I got it!')
        return 1
    else:
        return 1 + guess( hidden )
```

Name(s):

Random "Quiz"

1/1	4/7
2/4	3/8
3/7	3/9

how likely is each?

```
from random import *
choice( [1,2,3,2] )
```

What's the most likely return value here?

[0,1,2,3,4]

```
choice( list(range(5))+[4,2,4] )
```

What's the most likely return value?

```
choice( list(range(7)) )
```

More likely <u>even</u> or <u>odd</u>? 0 is even!

Careful on these...

```
choice( '1,2,3,4' )
```

What's the most likely return value here?

```
choice( ['1,2,3,4'] )
```

What's the most likely return value here?

```
choice( '[1,2,3,4]' )
```

What's the most likely return value here?

```
uniform( -20.5, 0.5 )
```

What are the chances of this being > 0?

Syntax corner...

```
choice(0,1,2,3,4)

choice([list(range(5))])

choice[list(range(5))]
```

Which <u>**two**</u> of these 3 are *syntax errors*?

Also, what does the ***third*** one – the one syntactically correct – actually *do*?

Data is in black. **Probabilities are in blue.**

Team up and try this on the backpage first...

```
from random import *
choice( [1,2,3,2] )
```

What's the most likely return value here? 2 **2/4**

probabilities in blue...

[0,1,2,3,4]

```
choice( list(range(5))+[4,2,4,2] )
```

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

What's the most likely return value? 4 **3/8**

```
choice( list(range(7)) )
```

[0,1,2,3,4,5,6]

More likely even or odd? 0 is even! *even* **4/7**

Careful on these...

```
choice( '1,2,3,4' )
```

What's the most likely return value here? ',' **3/7**

```
choice( ['1,2,3,4'] )
```

What's the most likely return value here? '1,2,3,4' **1/1**

```
choice( '[1,2,3,4]' )
```

What's the most likely return value here? ',' **3/9**

```
uniform( -20.5, 0.5 )
```

What are the chances of this being > 0? **1/42**

```
choice(0,1,2,3,4)
```

syntax error: needs list [...] or str '...'

```
choice([list(range(5))])
```

correct: ***always* returns [0,1,2,3,4]**

1/1 chance

```
choice[list(range(5))]
```

syntax error: needs parens: choice(...)

Data is in black. Probabilities are in blue.

```
from random import *
choice( [1,2,3,2] )
```

What's the most likely return value here? 2

2/4

Team up and try this on the backpage first...

probabilities in blue...

[0,1,2,3,4]

```
choice( list(range(5))+[4,2,4,2] )
```

What'...

3/8

c

Care

4/7

c

3/7

ch

1/1

ch

3/9

un

Pass these eastward!

```
cho
```

syntax error: needs list [...] or str '...'

```
choice([list(range(5))])
```

correct: *always* returns [0,1,2,3,4]

1/1 chance

```
choice[list(range(5))]
```

syntax error: needs parens: choice(...)

The two *Monte Carlos*

and their denizens…

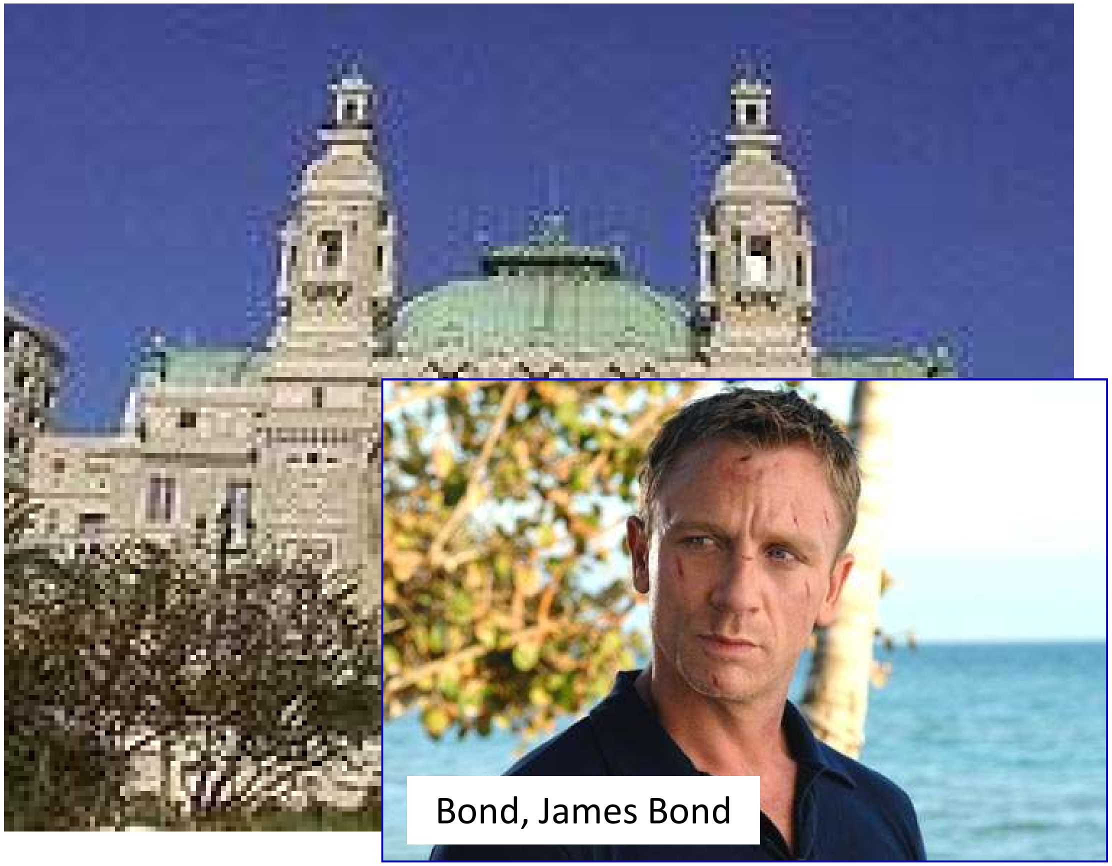
Monte Carlo casino, **Monaco**

Insights via *random trials*

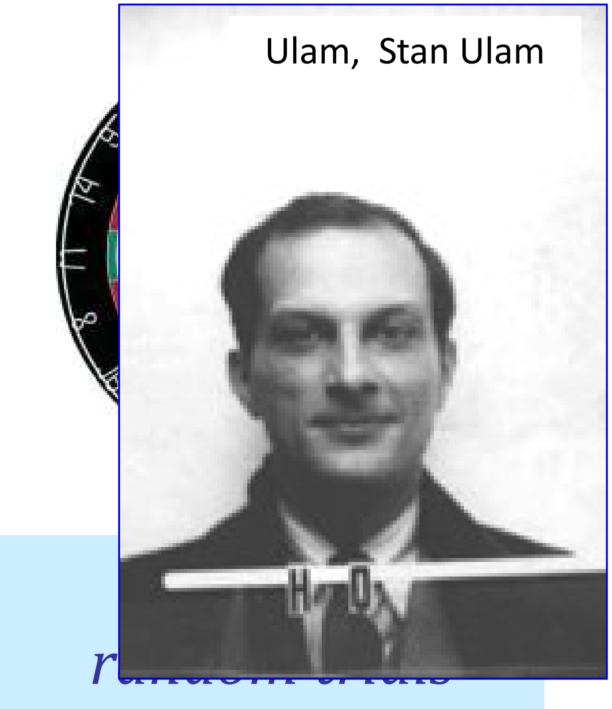
Monte Carlo methods, **Math/CS**

The two *Monte Carlos*

and their denizens…



Monte Carlo casino, **Monaco**



Monte Carlo
methods, **Math/CS**

Monte Carlo in action

How many doubles will you get in **N** rolls of 2 dice?

N is the total number of rolls

```
def countDoubles( N ):
    """  input: the # of dice rolls to make
        output: the # of doubles seen """
    if N == 0:
        return 0      # zero rolls, zero doubles…
    else:
        d1 = choice( [1,2,3,4,5,6] )
        d2 = choice( list(range(1,7)) )

        if d1 != d2:
            return 0+countDoubles( N-1 )   # not doubles
        else:
            return 1+countDoubles( N-1 )   # DOUBLES! Add 1
```

How are these the two dice?

where and how is the check for doubles being done?

Monte Carlo *Let's Make a Deal*…

Monte Carlo *Let's Make a Deal*...

'09-now

Monte Carlo *Let's Make a Deal*...

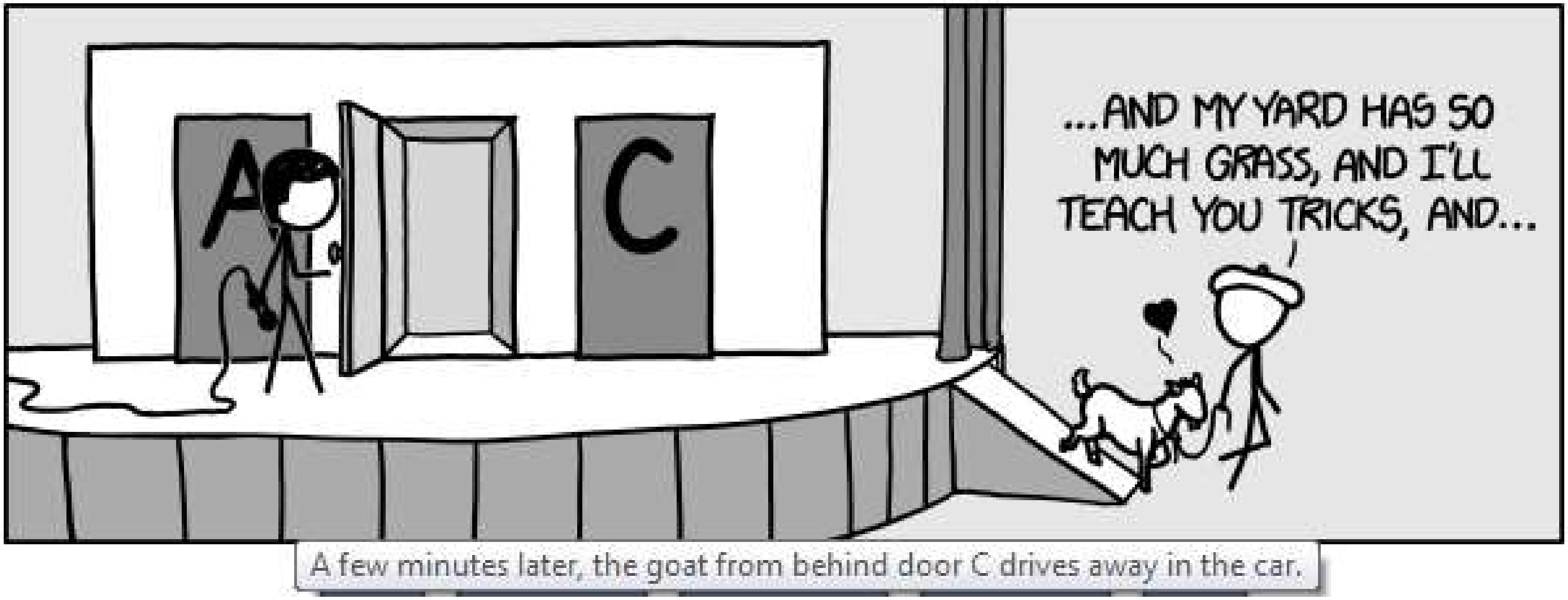
Monty Hall

'63-'86

inspiring the *Monty Hall paradox*

Let's make a deal: XKCD's take…

MONTY HALL


A
C
...AND MY YARD HAS SO MUCH GRASS, AND I'LL TEACH YOU TRICKS, AND...

A few minutes later, the goat from behind door C drives away in the car.

… what if you considered the goat the grand prize!?

Monte Carlo Monty Hall

Suppose you always **switch** to the other door...
What are the chances that you will win the prize ?

Let's play (randomly) 300 times and see!

Monte Carlo Monty Hall

'switch' or 'stay'

Your initial choice!

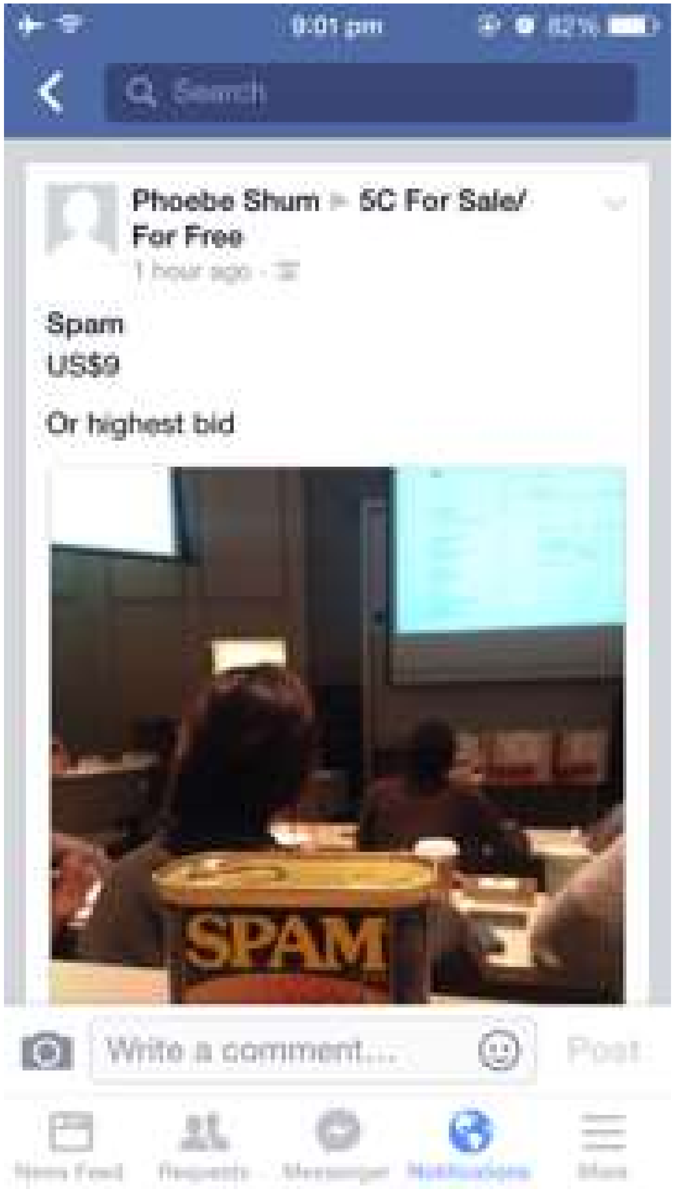
number of times to play

```
def MCMH( init, sors, N ):
  """ plays the "Let's make a deal" game N times
      returns the number of times you win the *Spam!*
  """
  if N == 0: return 0            # don't play, can't win
  przDoor = choice([1,2,3])      # where the spam (prize) is…

  if   init == przDoor and sors == 'stay':   result = 'Win!'
  elif init == przDoor and sors == 'switch': result = 'lose'
  elif init != przDoor and sors == 'switch': result = 'Win!'
  else:                                      result = 'lose'

  print 'Time', N, 'you', result

  if result == 'Win!':  return 1 + MCMH( init, sors, N-1 )
  else:                 return 0 + MCMH( init, sors, N-1 )
```

	0	1	2	3	4	5	6	7	8	9
A	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
B	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
C	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
D	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
E	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
F	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
G	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
H	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥

	0	1	2	3	4	5	6	7	8	9
A	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
B	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
C	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
D	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
E	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
F	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
G	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
H	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥

If you win some SPAM… ?   or pmfp... ?



Phoebe Shum ▸ 5C For Sale/ For Free

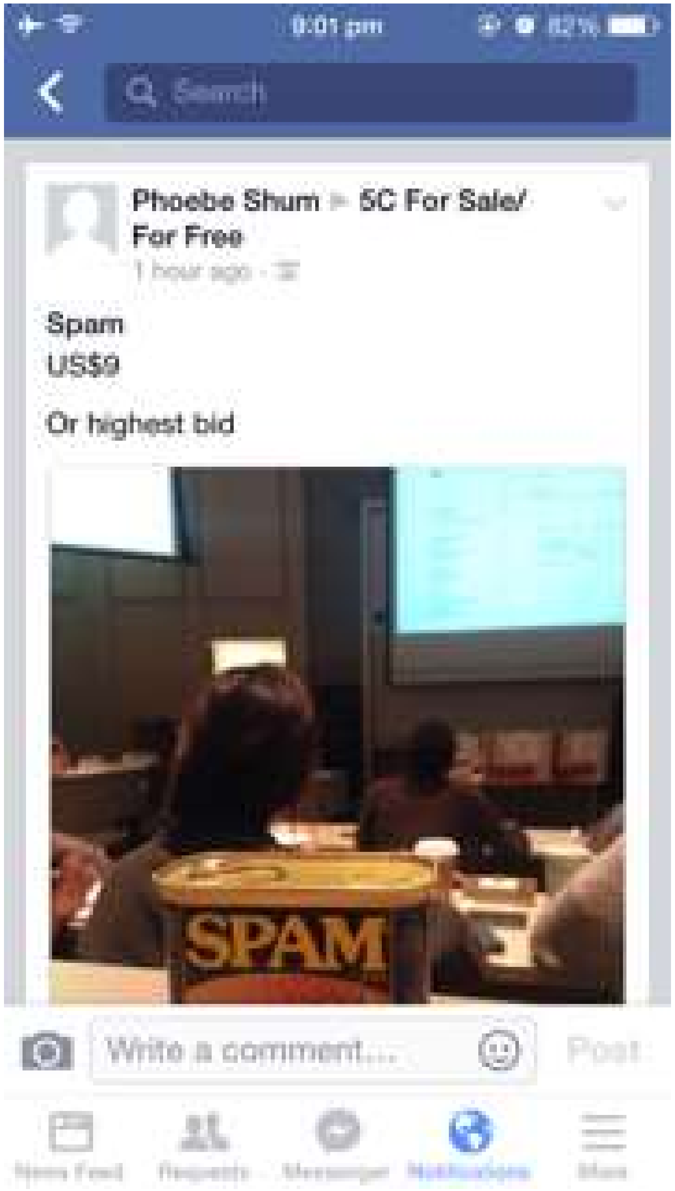
1 hour ago

Spam

US$9

Or highest bid

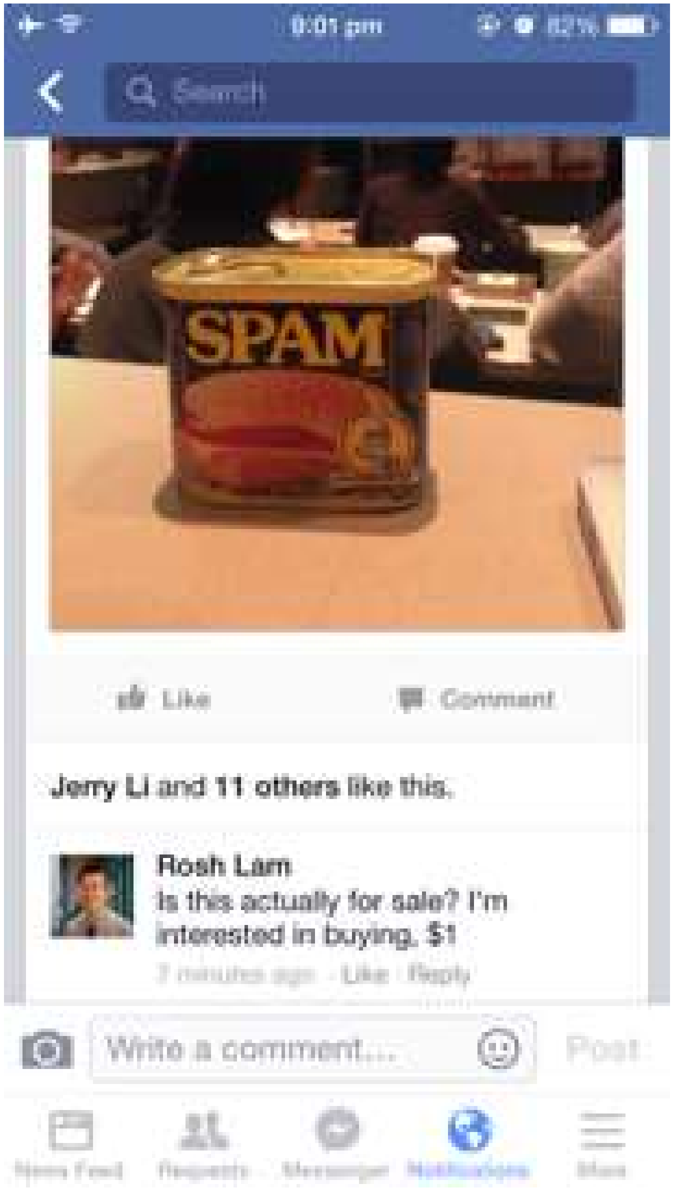
If you win some SPAM… ?   or pmfp… ?



Phoebe Shum ▸ 5C For Sale/ For Free
1 hour ago

Spam
US$9

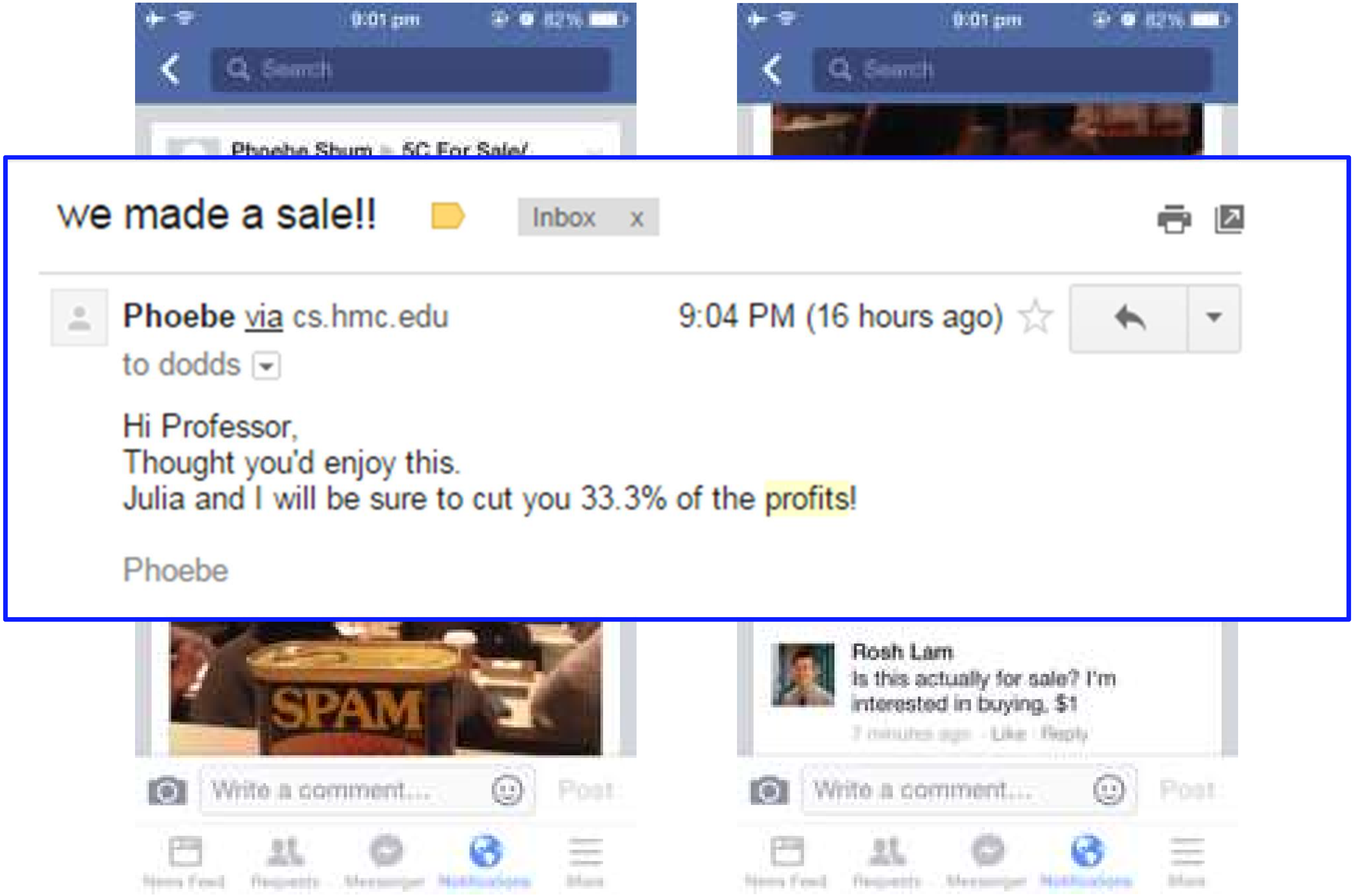
Or highest bid



Jerry Li and 11 others like this.

Rosh Lam
Is this actually for sale? I'm interested in buying. $1
7 minutes ago · Like · Reply

If you win some SPAM… ?   or pmfp… ?



Phoebe Shum ▸ 5C For Sale/

we made a sale!! Inbox x

Phoebe via cs.hmc.edu 9:04 PM (16 hours ago)

to dodds

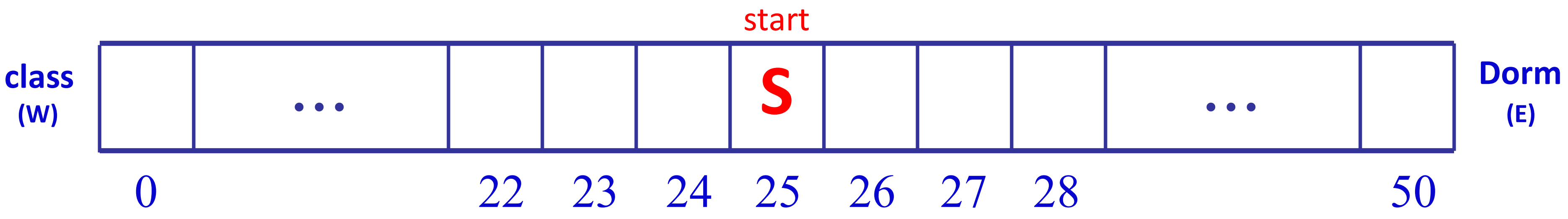
Hi Professor,
Thought you'd enjoy this.
Julia and I will be sure to cut you 33.3% of the profits!

Phoebe

Rosh Lam
Is this actually for sale? I'm interested in buying. $1
7 minutes ago · Like · Reply

An example *closer to home*

hw2pr2



An overworked 5C student **(S)** leaves H/S after their "late-night" breakfast – or lunch. Each moment, they randomly stumble toward class **(W)** or the dorm **(E)**

Once the student arrives at the dorm or classroom, the trip is complete.
The program should then print the total number of steps taken.

Write a program to model ***and analyze!*** this scenario...

`rwpos(st,nsteps)`

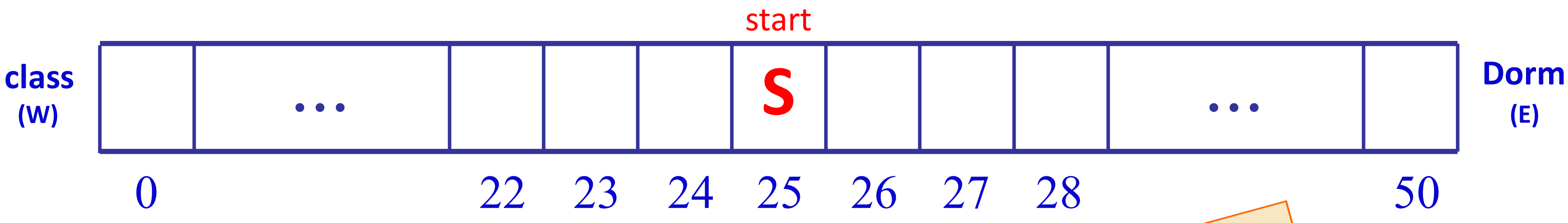
take **nsteps** random
steps starting at **st**

`rwsteps(st,low,hi)`

take random steps starting at **st**
until you reach either **low** or **hi**

An example *closer to home*

hw2pr2



An overworked 5C student **(S)** l… r
"late-night" breakfa… y
randoml… **(E)**

O… …n, the trip is complete.
…total number of steps taken.

Your task: To create this as an "ASCII" animation

Write …ogram to model ***and analyze!*** this scenario...

rwpos(st,nsteps)

take **nsteps** random
steps starting at **st**

rwsteps(st,low,hi)

take random steps starting at **st**
until you reach either **low** or **hi**

Lab 2 ~ *Python's Etch-a-Sketch*

Lab! *Python's Etch-a-Sketch*

No way this is real… but it is !

www.gvetchedintime.com

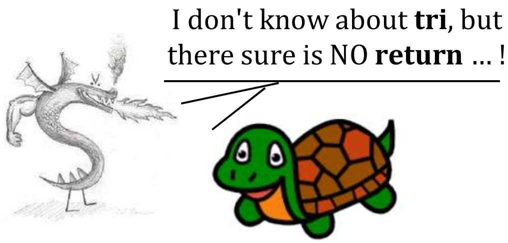
more *usual* etch-a-sketch work...

Single-path recursion

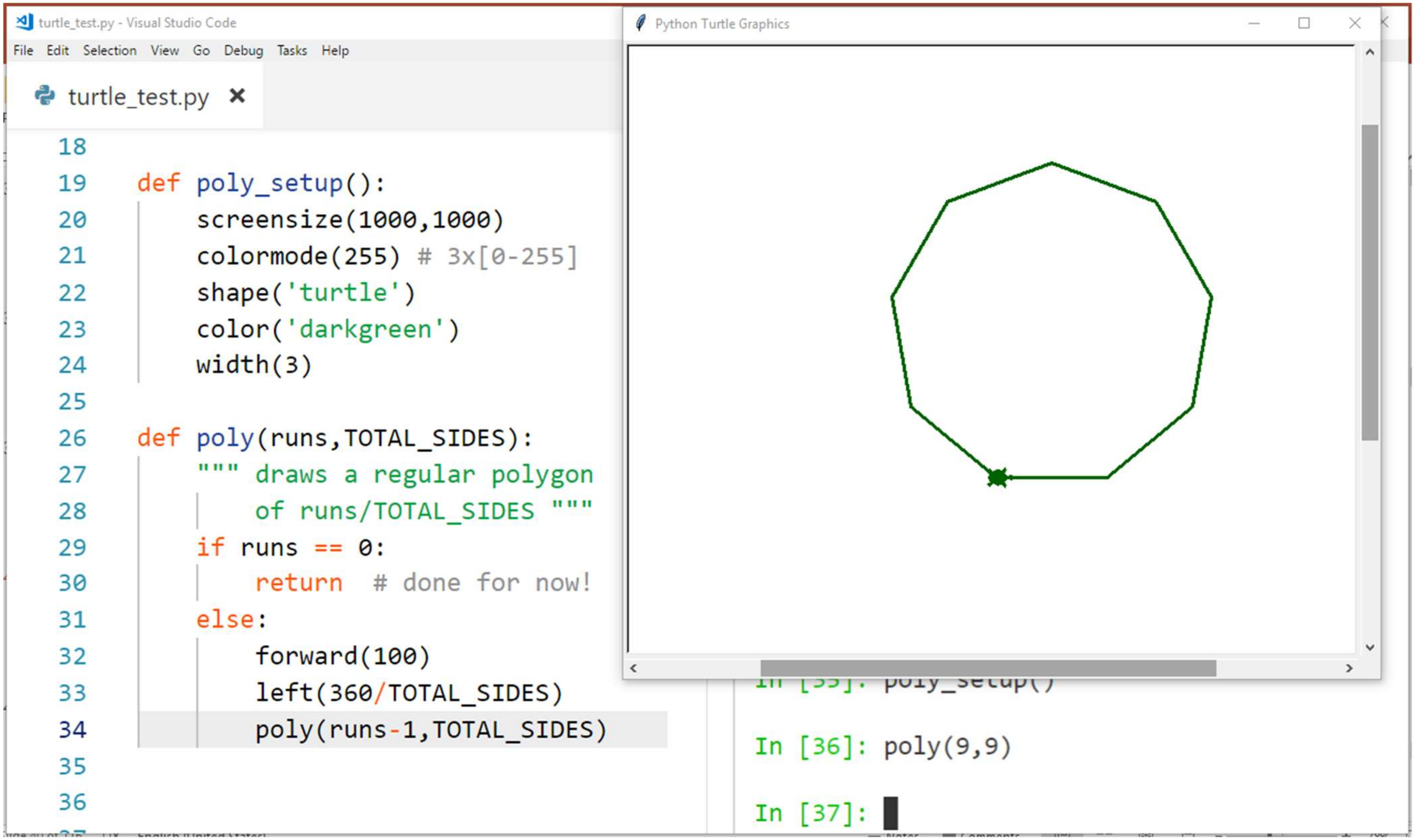
```
def tri():    # define it!
    """ a triangle!
    """
    forward(100)
    left(120)
    forward(100)
    left(120)
    forward(100)
    left(120)

# run
tri()
```

Let's **tri** this with recursion:

```
def tri( n ):
    """ draws a triangle """
    if n == 0: return
    else:
      forward(100) # one side
      left(120)    # turn 360/3
      tri( n-1 )   # draw rest
```


I don't know about **tri**, but
there sure is NO **return** ... !

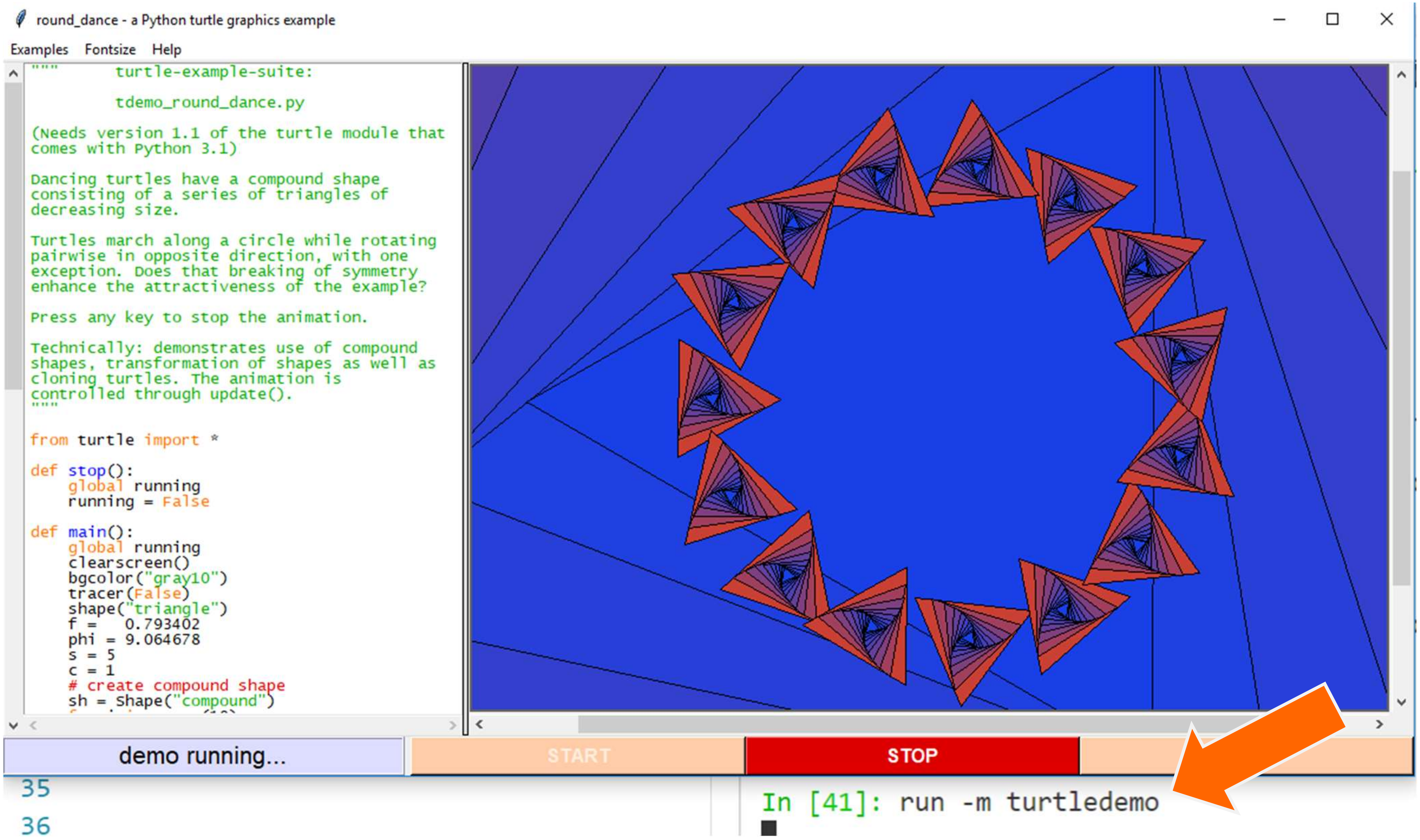
Turtle's ability?   It varies...


turtle_test.py - Visual Studio Code
File Edit Selection View Go Debug Tasks Help
turtle_test.py
Python Turtle Graphics

```python

def poly_setup():
    screensize(1000,1000)
    colormode(255) # 3x[0-255]
    shape('turtle')
    color('darkgreen')
    width(3)

def poly(runs,TOTAL_SIDES):
    """ draws a regular polygon
        of runs/TOTAL_SIDES """
    if runs == 0:
        return  # done for now!
    else:
        forward(100)
        left(360/TOTAL_SIDES)
        poly(runs-1,TOTAL_SIDES)


```

```
In [35]: poly_setup()

In [36]: poly(9,9)

In [37]: 
```

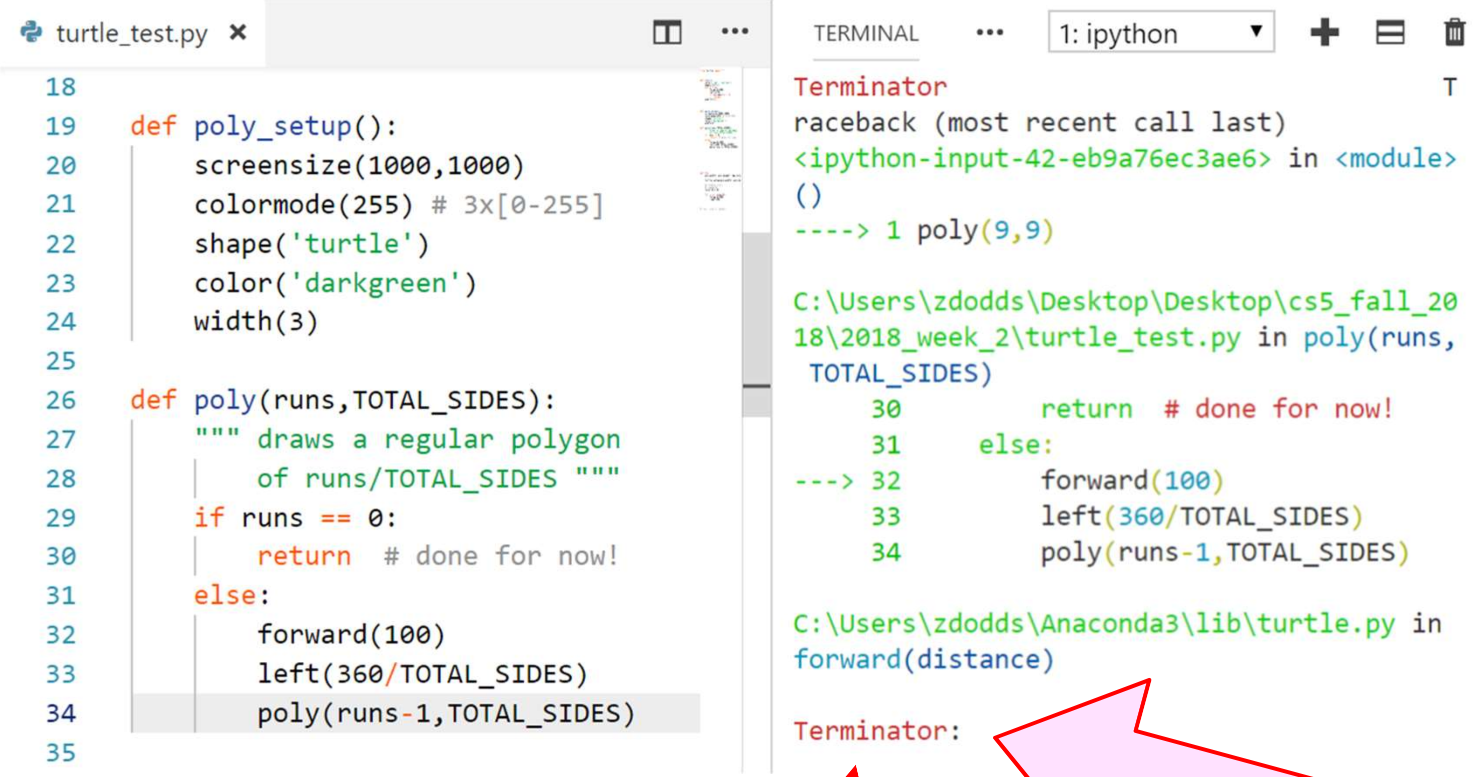
Turtle's ability?   It varies widely!



```
"""      turtle-example-suite:

         tdemo_round_dance.py

(Needs version 1.1 of the turtle module that
comes with Python 3.1)

Dancing turtles have a compound shape
consisting of a series of triangles of
decreasing size.

Turtles march along a circle while rotating
pairwise in opposite direction, with one
exception. Does that breaking of symmetry
enhance the attractiveness of the example?

Press any key to stop the animation.

Technically: demonstrates use of compound
shapes, transformation of shapes as well as
cloning turtles. The animation is
controlled through update().
"""

from turtle import *

def stop():
    global running
    running = False

def main():
    global running
    clearscreen()
    bgcolor("gray10")
    tracer(False)
    shape("triangle")
    f =   0.793402
    phi = 9.064678
    s = 5
    c = 1
    # create compound shape
    sh = Shape("compound")
```

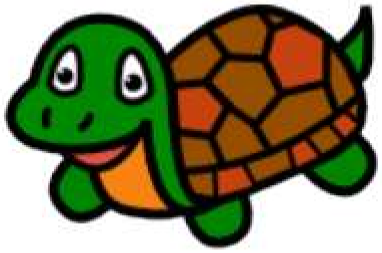
```
In [41]: run -m turtledemo
```

Warning: *Terminator error!*



```
def poly_setup():
    screensize(1000,1000)
    colormode(255) # 3x[0-255]
    shape('turtle')
    color('darkgreen')
    width(3)

def poly(runs,TOTAL_SIDES):
    """ draws a regular polygon
        of runs/TOTAL_SIDES """
    if runs == 0:
        return  # done for now!
    else:
        forward(100)
        left(360/TOTAL_SIDES)
        poly(runs-1,TOTAL_SIDES)
```

```
Terminator                                   T
raceback (most recent call last)
<ipython-input-42-eb9a76ec3ae6> in <module>
()
----> 1 poly(9,9)

C:\Users\zdodds\Desktop\Desktop\cs5_fall_20
18\2018_week_2\turtle_test.py in poly(runs,
 TOTAL_SIDES)
     30             return  # done for now!
     31         else:
---> 32             forward(100)
     33             left(360/TOTAL_SIDES)
     34             poly(runs-1,TOTAL_SIDES)

C:\Users\zdodds\Anaconda3\lib\turtle.py in
forward(distance)

Terminator:
```



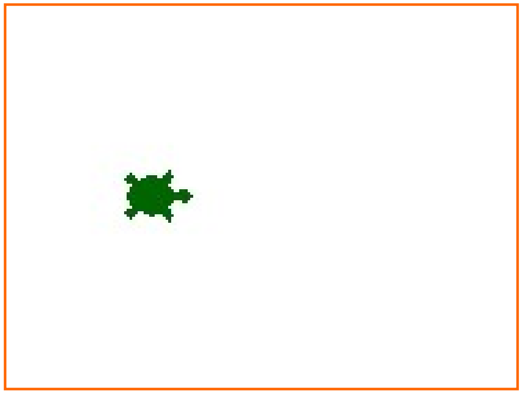
Problem: *Terminator Error*

Solution: *Just run it again!*

-- just call me again

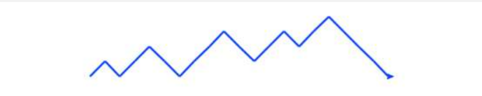
Be the turtle !

(1) What would **chai(100)** draw?



```
def chai(dist):
    """ mystery fn! """
    if dist < 5: return

    forward(dist)
    left(90)
    forward(dist/2.0)
    right(90)
    # recurse here?
    right(90)
    forward(dist)
    left(90)
    # recurse here?
    left(90)
    forward(dist/2.0)
    right(90)
    backward(dist)
```

(2)



one possible result of **rwalk(20)**

Have **rwalk** draw a "stock-market" path of **N** steps of 10 pixels each. *Use recursion.*

```
from random import *

def rwalk(N):
    """ make N 10-pixel steps, NE or SE """

    if N == 0: return

    elif choice(['left','right']) == 'left':

        left(45)
        forward(10)
                              ⇦ ?

    else:  # this handles 'right'

                                   ⇦ ?
```

Extra! How could you make this a bull (or a bear) market?

Extra #2! What if the line chai(dist/2) were placed between the two right(90) lines? And/or between the two left(90) lines?

rwalk(N) is a random "stock market" walk...

```
from random import *

def rwalk(N):
    """ make N 10-px steps, NE or SE """
    if N == 0:     return

    elif choice(['left','right'])=='left':
        left(45)
        forward(10)
        right(45)
        rwalk( N-1 )

    else:  # 'right'
        right(45)
        forward(10)
        left(45)
        rwalk( N-1 )
```

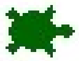
What if we ***didn't*** turn back to face east each time?

"Single-path" (or counting) recursion

Single-path recursion

What does **chai(100)** do here?

```
def chai(dist):
    """ mystery! """
    if dist<5:
        return

    forward(dist)
    left(90)
    forward(dist/2.0)
    right(90)

    right(90)
    forward(dist)
    left(90)

    left(90)
    forward(dist/2.0)
    right(90)
    backward(dist)
```

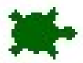


How could you add more to each T's tips?

Why are there two identical commands in a row ~ twice!?

Branching recursion

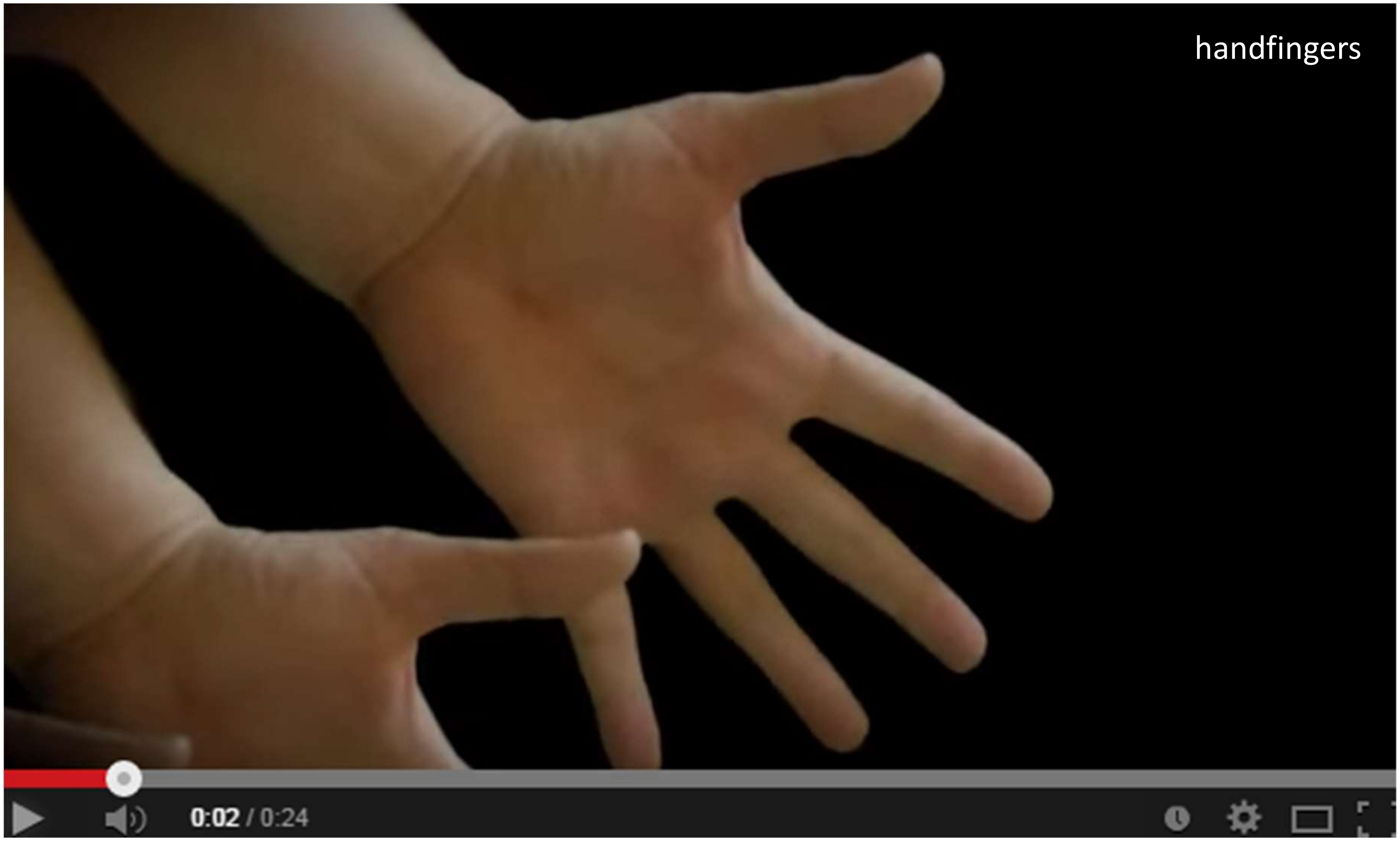
Now, what does **`chai(100)`** do?



```
def chai(dist):
    """ mystery! """
    if dist<5:
        return

    forward(dist)
    left(90)
    forward(dist/2.0)
    right(90)
    chai(dist/2)
    right(90)
    forward(dist)
    left(90)
    chai(dist/2)
    left(90)
    forward(dist/2.0)
    right(90)
    backward(dist)
```

"Multiple-path" (or branching) recursion

Cyriak: ***conceptually disruptive*** recursion…

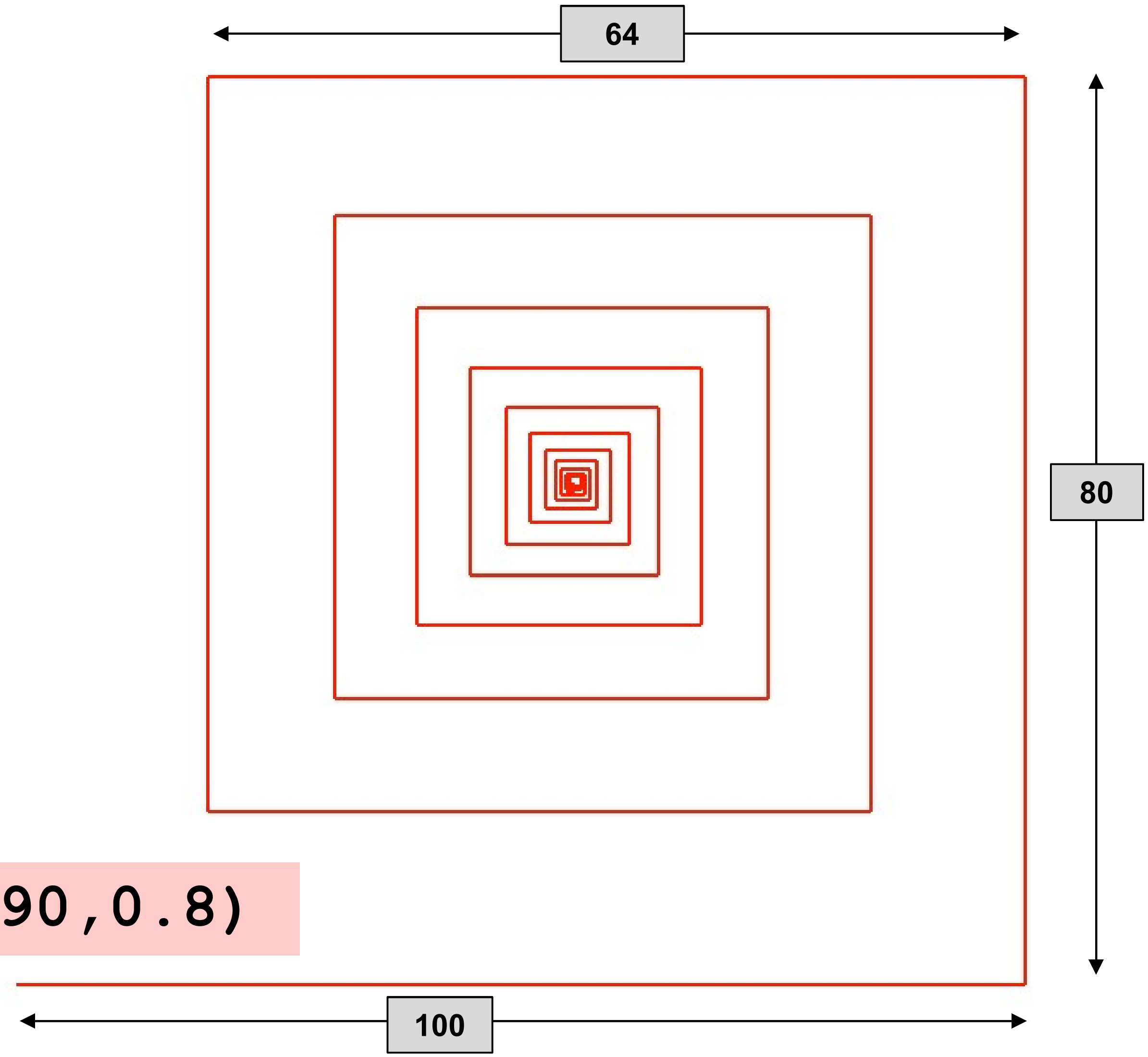


is the ***branching***, not the ***single-path*** variety.

Single-path or *Branching* *recursion here?*

lab ~ hw2pr1

fractal art



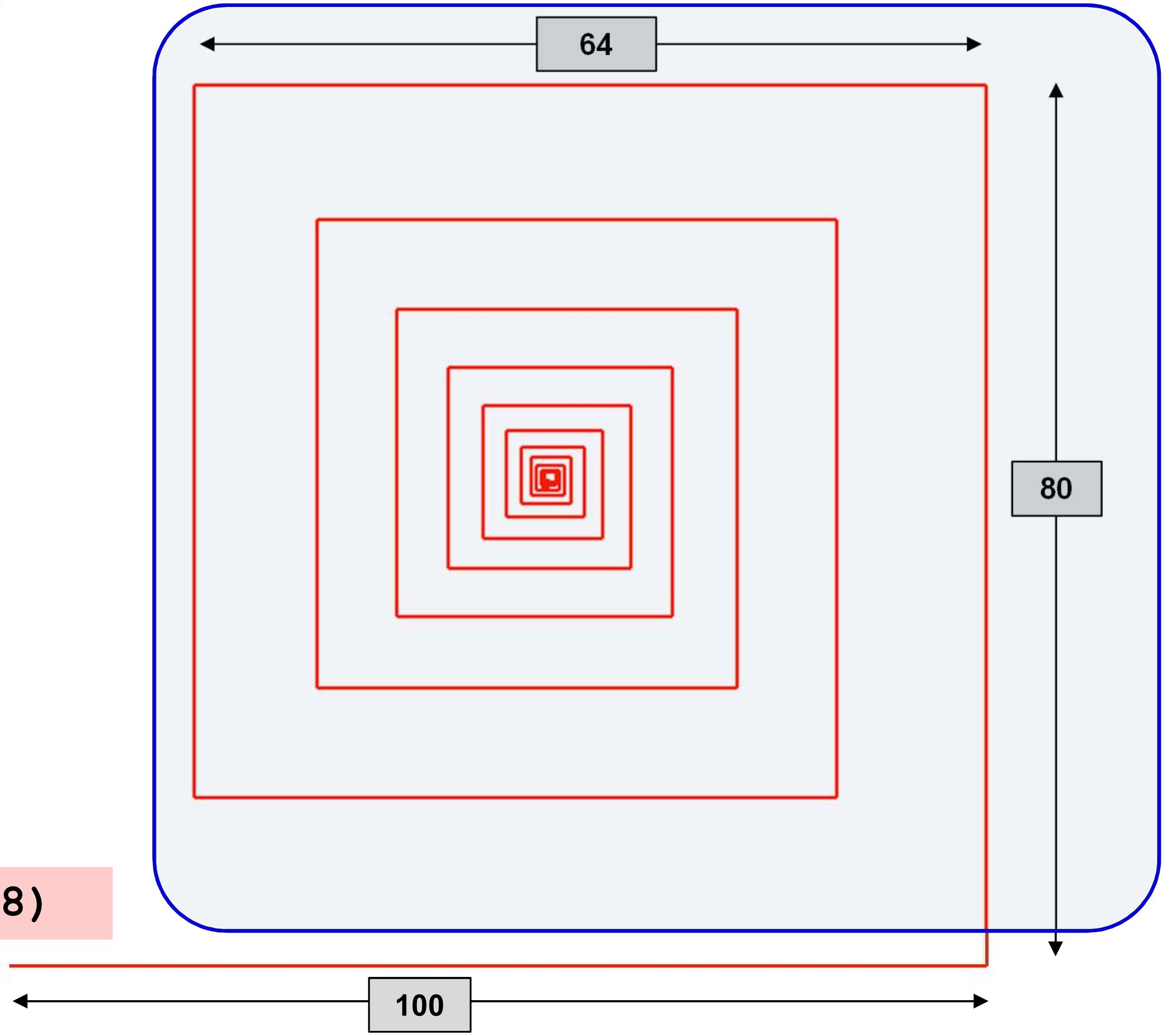
`spiral(100,90,0.8)`

`spiral( initLength, angle, multiplier )`

lab ~ hw2pr1

fractal art

`spiral(80,90,0.8)`

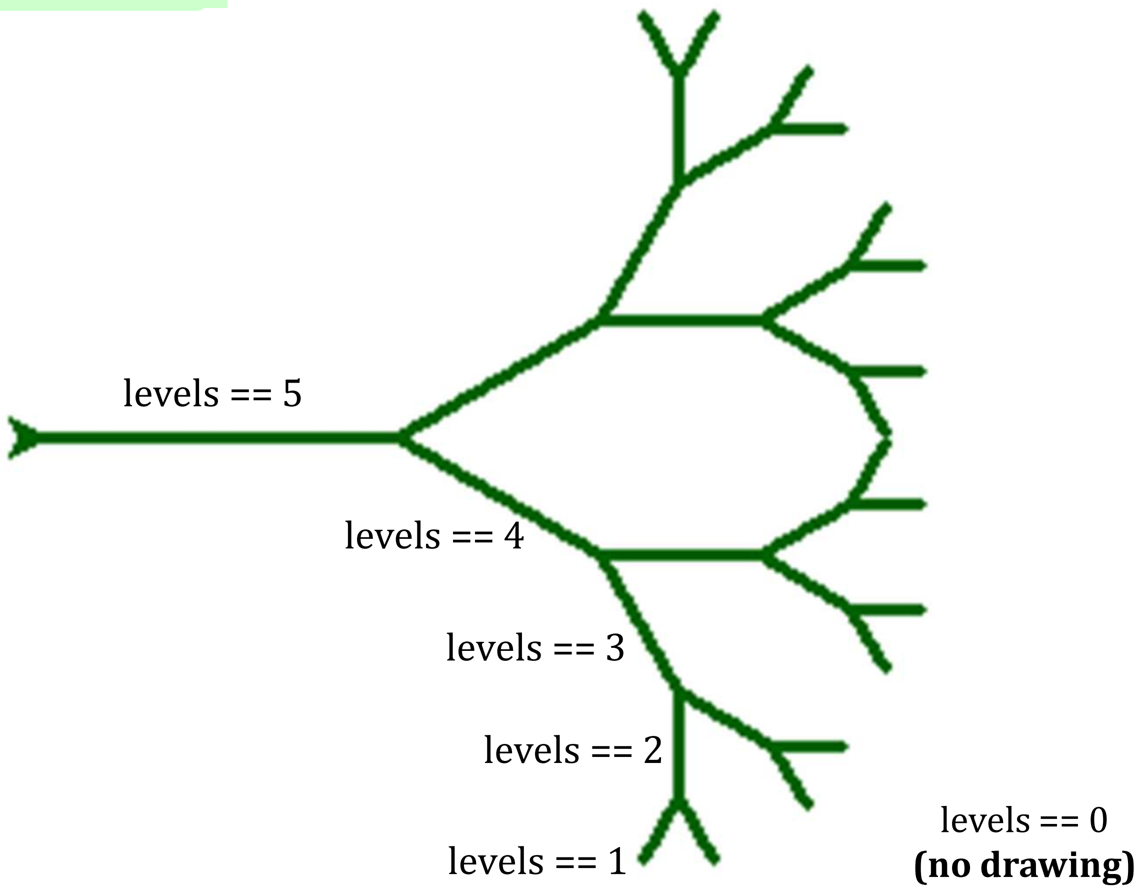


`spiral(100,90,0.8)`

```
spiral( initLength, angle, multiplier )
```

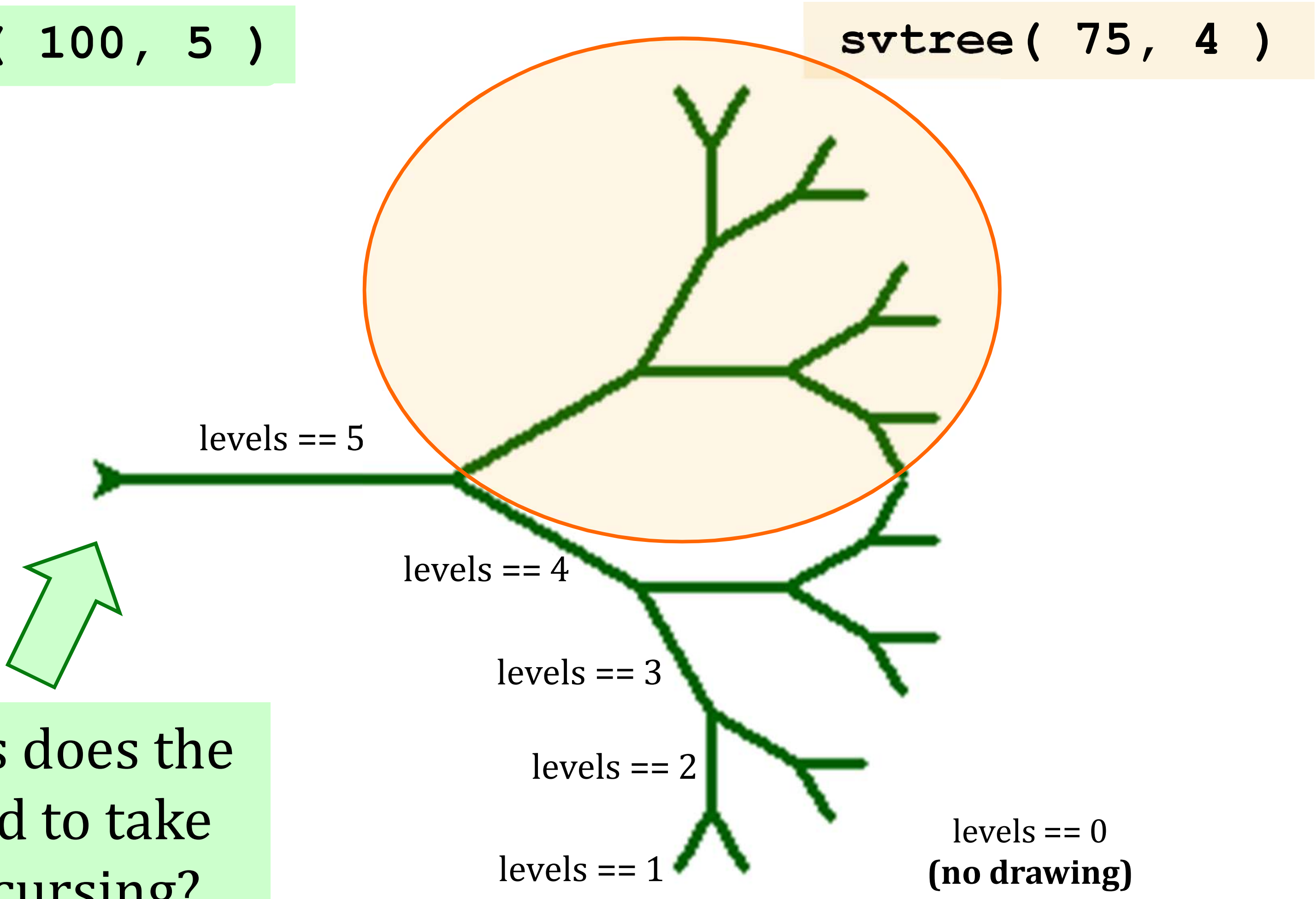
svtree(trunkLength, levels)

```
svtree( 100, 5 )
```



***Single-path* or *Branching* recursion here?**

```
svtree( trunkLength, levels )
```

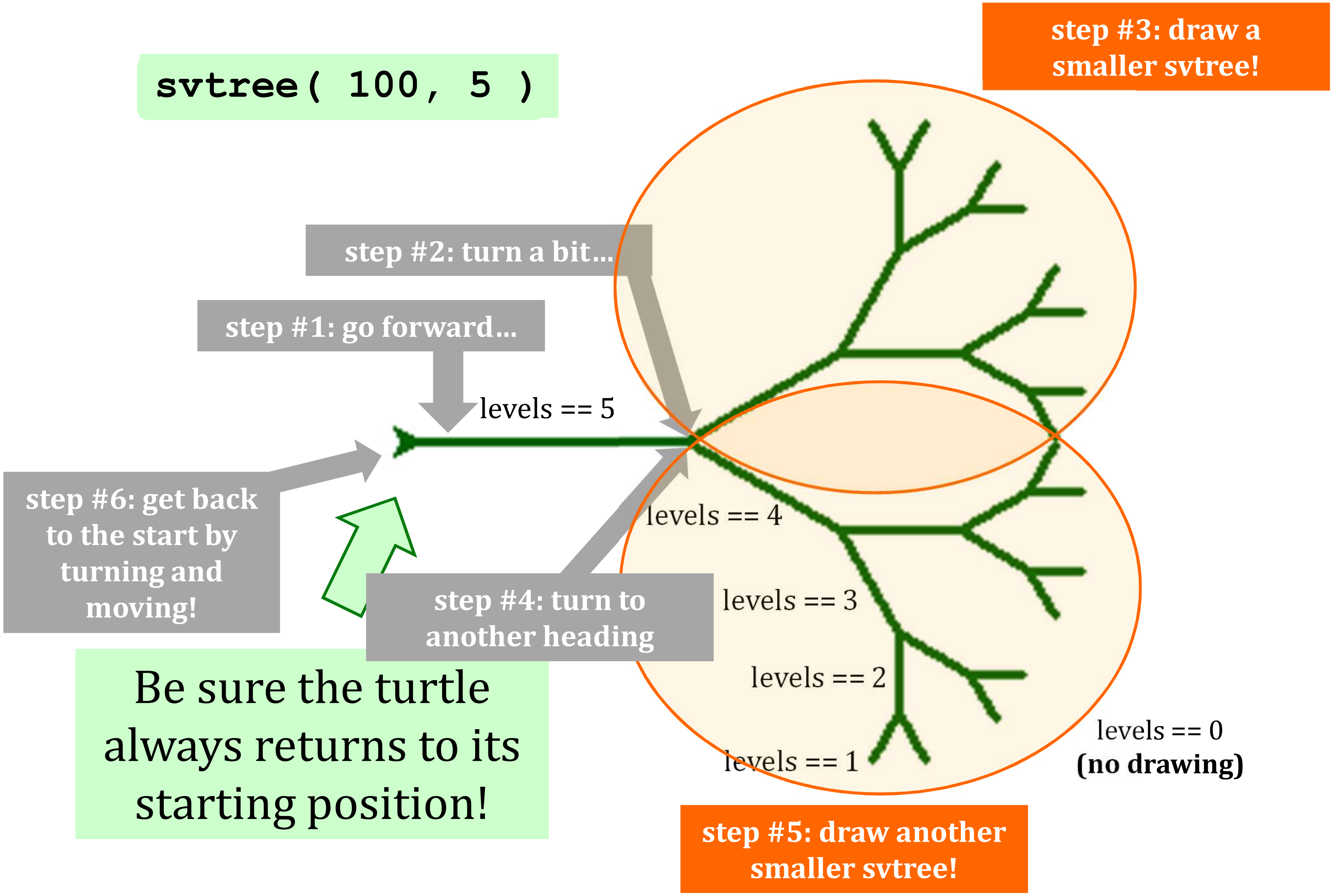


What steps does the turtle need to take before recursing?

Branching recursion!

`svtree( trunkLength, levels )`

```
svtree( 100, 5 )
```



step #1: go forward...

levels == 5

step #2: turn a bit...

step #3: draw a smaller svtree!

step #4: turn to another heading

levels == 4

levels == 3

levels == 2

levels == 1

levels == 0
(no drawing)

step #5: draw another smaller svtree!

step #6: get back to the start by turning and moving!

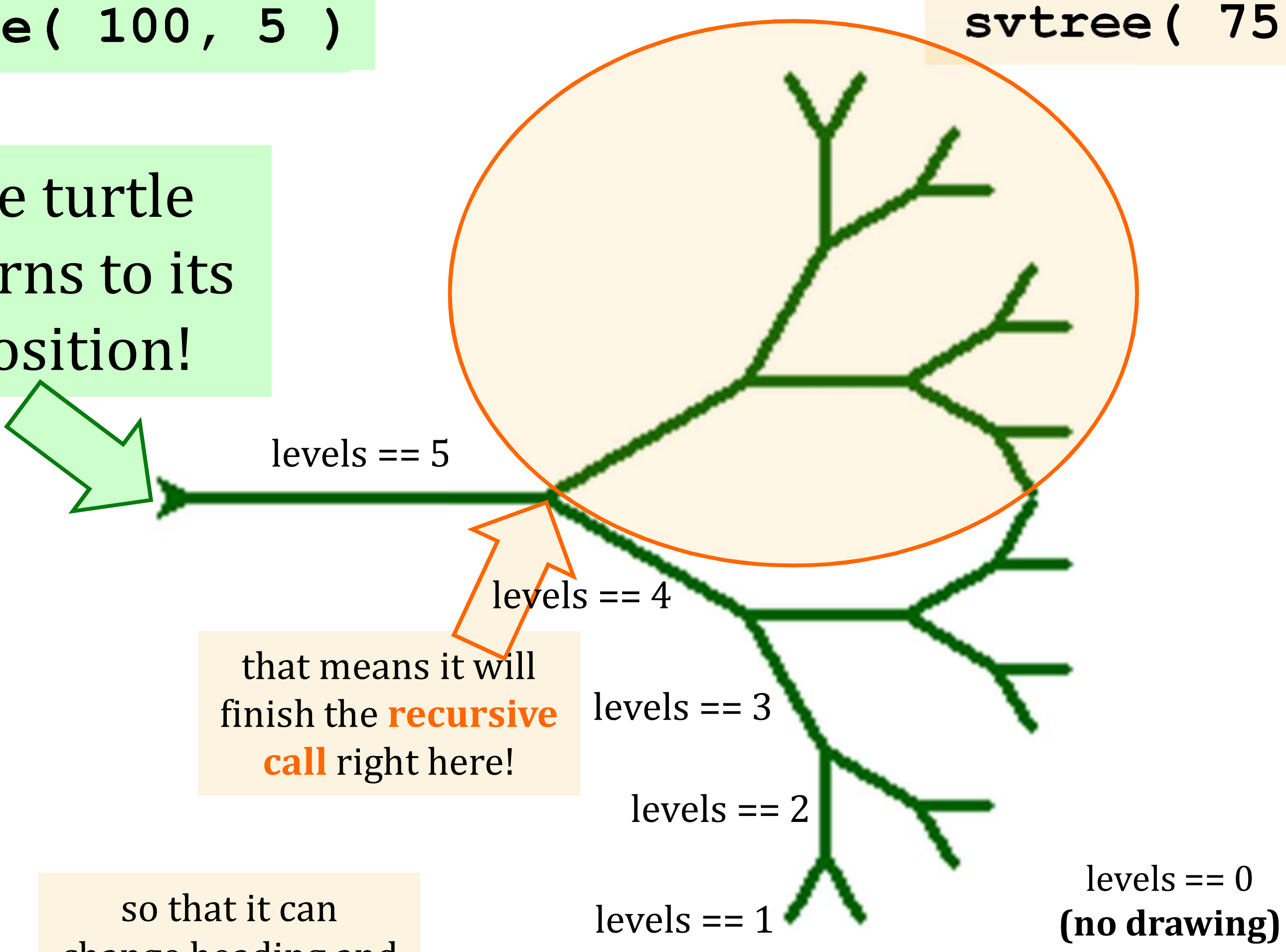
Be sure the turtle always returns to its starting position!

Branching ***recursion!***

```
svtree( trunkLength, levels )
```

```
svtree( 100, 5 )
```

Be sure the turtle always returns to its starting position!



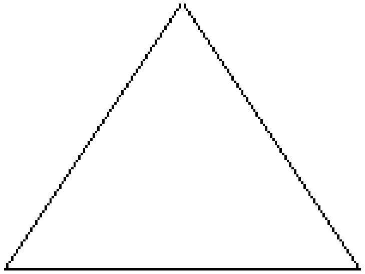
```
svtree( 75, 4 )
```

that means it will finish the **recursive call** right here!

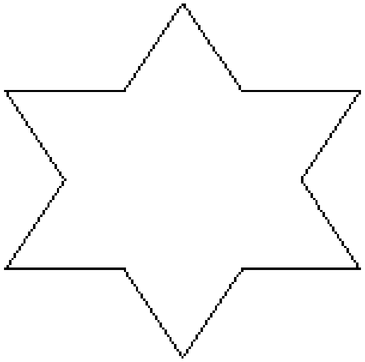
so that it can change heading and draw another one...

***Branching* recursion!**

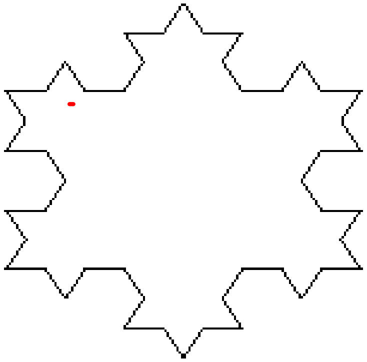
The Koch curve



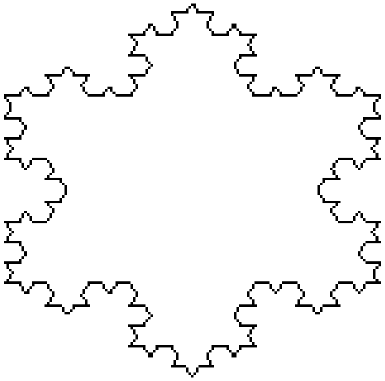
`snowflake(100, 0)`



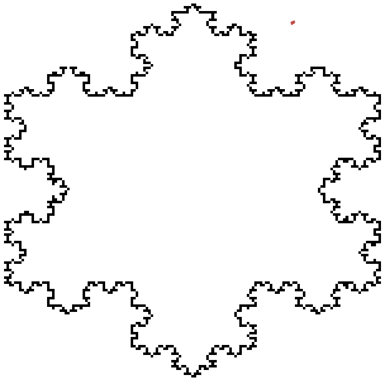
`snowflake(100, 1)`



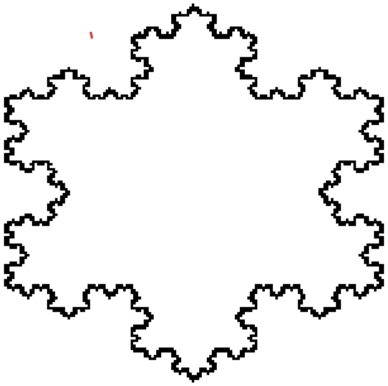
`snowflake(100, 2)`



`snowflake(100, 3)`



`snowflake(100, 4)`

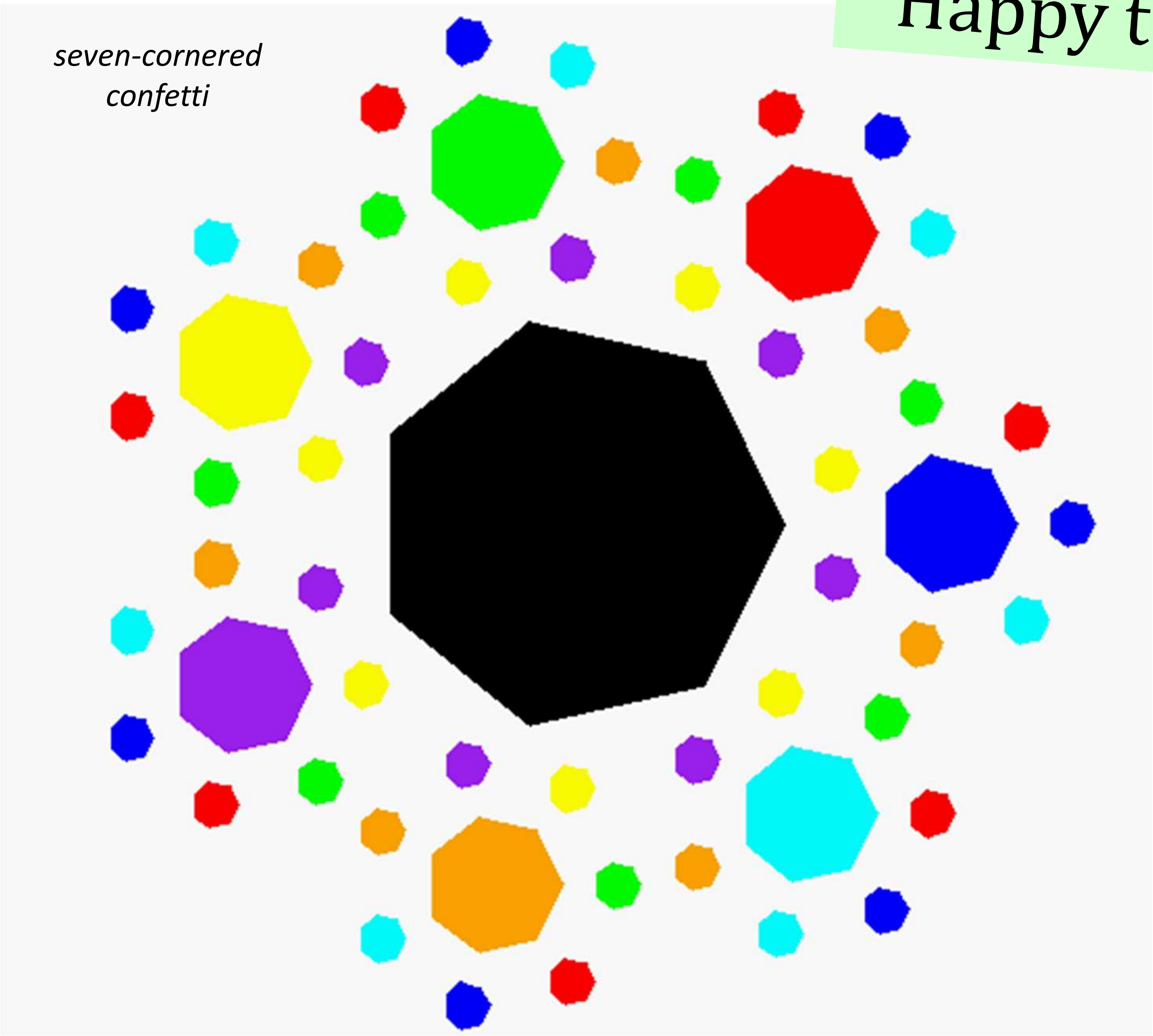


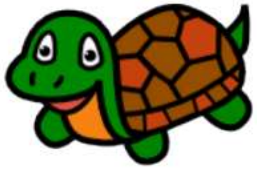
`snowflake(100, 5)`

***Single-path** or **Branching** recursion here?*

Recursive art?   Create your own...

Happy turtling in lab!





What? This is too happy to be art... My recursive compositions burninate even Cyriak's brain!