

Cyriak: *conceptually disruptive* recursion...

Baaa

Fractals and Turtles



CS 5
alien on
strike!

CS 5 green mascot
representing today's
terrestrial themes



CS 5 Today

hw2 due soon~ish

Lots of tutoring...



How random!

Cyriak: *conceptually disruptive* recursion...

Baaa

Fractals and Turtles



CS 5
alien on
strike!

CS 5 green mascot
representing today's
terrestrial themes



Applications!

CS 5

hw2 due soon~ish

Lots of tutoring...

COWS & COWS & COWS

How random!

The key to understanding recursion is, first, to understand recursion.

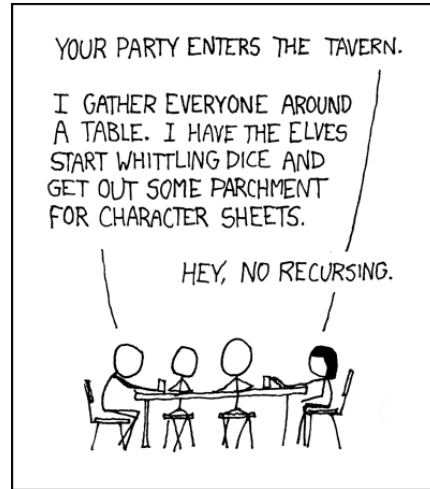
- former CS 5 student

```
def cs5(hw) :  
    if hw == 0:  
        return "Graded!"  
    elif hw == 1:  
        return "Submitted!"  
    else:  
        return "Working! " + cs5(hw-1)
```

tutors @ McGregor: T/W/Th/F/Sa/Su/Mon.

Recursion's advantage:

It handles arbitrary structural depth – *all at once + on its own!*



As a hat, I'm recursive, too!

<https://www.youtube.com/watch?v=ybX9nVLtNi4> @ 0:08
<https://www.youtube.com/watch?v=8PhiSSnaUKk> @ 1:11

Pomona Sends Survey To Students To Find Out Why They Don't Take Surveys

Ima Firstyear

Declining survey response rates at Pomona College prompted the administration to send students a new survey this week, which will assess students' previous survey experiences and their survey preferences in hopes of explaining—and reversing—the decline.

"We know Pomona students have strong opinions about their education and their campus," said Vice President and Dean of Students Miriam Feldblum. "But what we find is that when we

offer students a chance to express those opinions via a general survey, we don't get as many responses as we expect. We want to know why, and that's why we're sending out this survey."

Students will be asked to self-identify at the start of the survey as a 'frequent responder,' 'occasional responder' or 'forgot the password to my Pomona webmail account three months ago.' According to Feldblum, these categories will help the administration create new strategies to engage more of the student population in responding to surveys.

The survey also addresses questions of methodology, incentive and access. It asks students to rank their preferences of survey provider, such as SurveyMonkey, Qualtrics and Google Forms, and to name their ideal survey prizes. It also asks students whether they would be more inclined to take school surveys via email, an iPhone app or voting machines in the dining halls complete with 'I Surveyed!' stickers.

Erika Bennett PO '17 said she found some of the questions confusing.

"I had to pick my favorite as-

essment scale," she said. "I had to rank 'Scale of one to five,' 'Strongly Disagree to Strongly Agree' and 'Sad Face to Happy Face' from least to most intuitive. But I'm not sure I did it correctly."

Bennett added that she did appreciate the chance to critique previous surveys.

"Just last month I took a survey with no progress bar at the bottom of each page," she said. "I felt lost and confused. I'm glad there's a real See SURVEY page 2



Are surveys the broccoli of our digital age?

Week 1, big-picture...

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

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

`3*4 +`

`2*7 +`

`4*4`

Sequential design...

Week 1, big-picture...

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

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

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

Recursive design...

Recursion's idea:

We handle EMPTY + 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
```

*Empty
Cases*

*Base
Cases*

Recursion's idea:

We handle EMPTY + FIRST

Recursion handles the REST

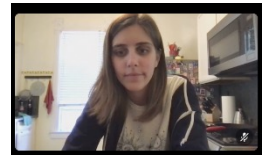
```
def dot( L, K ) :  
    if len(L) == 0 or len(K) == 0:  
        return 0.0
```

*Empty
Cases*

*Base
Cases*

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

*Specific/General
case(s)*



```
else:
```

combine

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

handle the
FIRST of **L**

handle the
FIRST of **K**

handle the first

handle the
REST of **L**

handle the
REST of **K**

recurse w/the rest

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]

$3*4 + \text{dot}([2,4], [7,4])$

L = [2,4] and K = [7,4]

$2*7 + \text{dot}([4], [4])$

L = [4] and K = [4]

$4*4 + \text{dot}([], [])$

L = [] and K = []

0.0

16.0

30.0

42.0

slow and steady!



Python 3.6

```
1 def dot( L, K ):
2     if len(L) == 0 or len(K) == 0:
3         return 0.0
4     if len(L) != len(K):
5         return 0.0
6     else:
7         return L[0]*K[0] + dot(L[1:],K[1:])
8
9
10 print(dot([3,2,4],[4,7,4]))
```

[Edit this code](#)

that has just executed

at line to execute

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

<< First

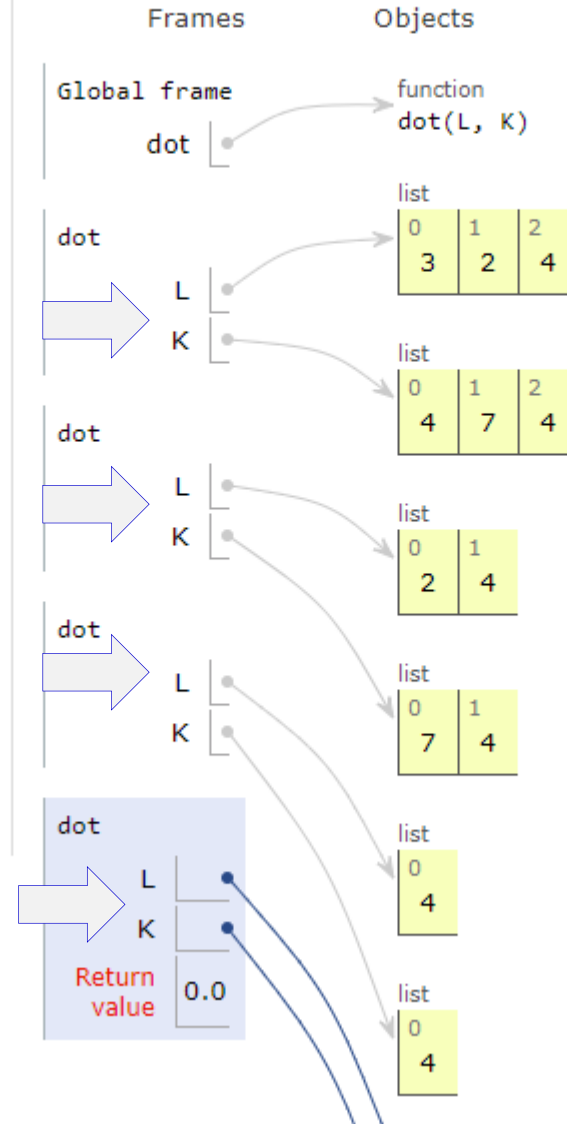
< Back

Step 18 of 21

Forward >

Last >>

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



Seeing the "stack" ...

single-path recursion

A random aside...

Libraries!

A random aside...

Libraries!

```
from random import *
```

```
choice( L )
```

don't need to use the library name

all random functions
are now available!

```
import random
```

requires using the
library name

```
random.choice( L )
```

A random aside...



Which Bender?

Which Dorm?

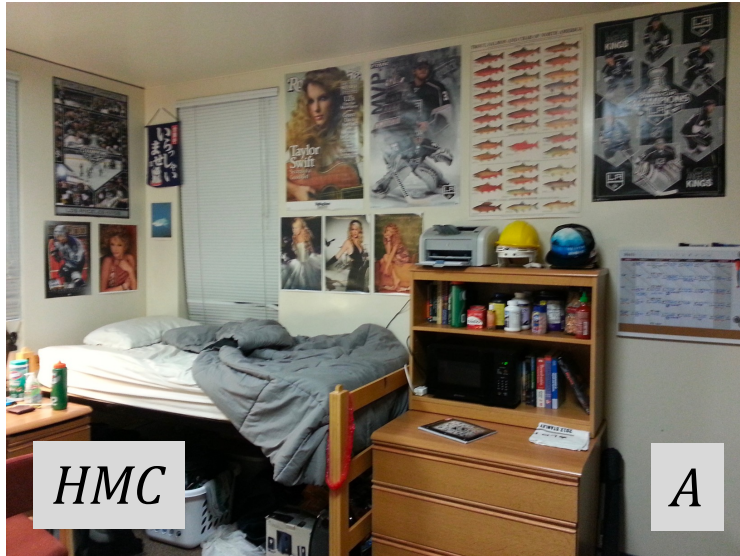


talk about random... !

School rivalries?

Which four Cs?

Which is which?



A random aside...

```
from random import *
```

all random functions are now available!

```
choice( L )
```

chooses 1 element from the sequence L

```
choice([ 'Zuko' , 'Katara' , 'Aang' , 'Appa' ])
```

```
choice([ 'sontag' , 'case' , 'linde' , 'atwood' ])
```

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

... or 1 character from a string

A random aside...

```
from random import *
```

all random functions are now available!

```
choice( L )
```

chooses 1 element from the sequence L

```
range(1,5) → [1,2,3,4]
```

```
range(5) → [0,1,2,3,4]
```

range

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

A *random* aside...

```
from random import *
```

all random functions are now available!

```
choice( L )
```

chooses 1 element from the sequence L

```
range(1,5) → [1,2,3,4]
```

```
range(5) → [0,1,2,3,4]
```

range

```
choice(range(100))
```

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

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

a random **float** from low to hi

```
In [23]: uniform(41,43)  
Out[23]: 42.00092480157684
```

floats have **16** places of precision

*Aargh –
so close!*



Using randomness
to our advantage:

"Monte Carlo Methods"

Randomness: Recursion-as-*"the future"*

```
def guess( hidden ) :  
    """ tries to guess our hidden number """  
    compguess = choice( 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

```
from random import *
import time

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

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

    if compguess == hidden: # at last!
        return 1
    else:
        return 1 + guess( hidden )
```

code available
in hw2pr3

Name: _____

1/1

4/7

2/4

3/8

3/7

3/9

how likely?

`choice([1,2,3,2])` —

What's the most likely return value here?

[0,1,2,3,4]

most likely value:

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

What's most likely?

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

`choice(list(range(7)))` —

More likely even or odd?

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

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

Watch out!

`choice`

`choice`

`uniform`

`choice`

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

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

Let's try it!

being > 0?

Syntax corner...

Which 2 of these 3 are *syntax errors*?

And what does the **third** one – the one that's syntactically correct – actually *do*?

Calling `uniform(41, 43)` 1000 times/s, how long until you obtain exactly 42.000000000000000?

Mind Muddler!

A float has 16 places of precision total. (14 places after the 42.)

Name: _____

1/1

4/7

~~2/4~~

~~3/8~~

3/7

3/9

how likely?

`choice([1,2,3,2])` —

What's the most likely return value here?

most likely value:

2

2/4

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

What's most likely?

4

3/8

`choice([0,1,2,3,4,4,2,4] list(range(7)))`

More likely even or odd? (0 is even)

odd or even?

4/7

`choice('1,2,3,4')` —

What's the most likely return value here?

2

2/7

`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?

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

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

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

Which 2 of these 3 are *syntax errors*?

And what does the **third** one – the one that's syntactically correct – actually *do*?

Syntax corner...

Calling `uniform(41,43)` 1000 times/s, how long until you obtain exactly 42.000000000000000?

Mind Muddler!

A float has 16 places of precision total. (14 places after the 42.)

Watch out!

Solutions!

choice

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

1/1

4/7

2/4

3/8

3/7

3/9

how likely?

What's the most likely return value here?

2

2/4

most likely value:

[0,1,2,3,4]

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

What's most likely?

4

3/8

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

choice (list(range(7)))

More likely even or odd? (0 is even)

even

4/7

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

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 '...'

Syntax corner...

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

Which 2 of these 3 are syntax errors?

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

syntactically correct - actually *do*?

1/1 chance

choice [list(range(5))]

syntax error: needs parens: choice[...]

Calling uniform(41,43) 1000 times/s, how long until you obtained exactly 42.00000000000000?

6,337 years!

and Muddler!

In [1]: 2*10**14/(1000*60*60*24*365.25) Out[1]: 6337.61756280579 float has 16 places of precision, total. (14 places after the 42.)

Watch out!

The two *Monte Carlos*

and their denizens...



Monte Carlo casino, Monaco



Insights via
random trials

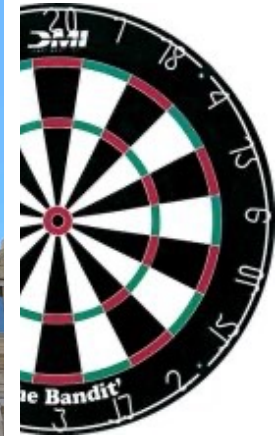
Monte Carlo
methods, Math/cs

The two *Monte Carlos*

and their denizens...



Monte Carlo

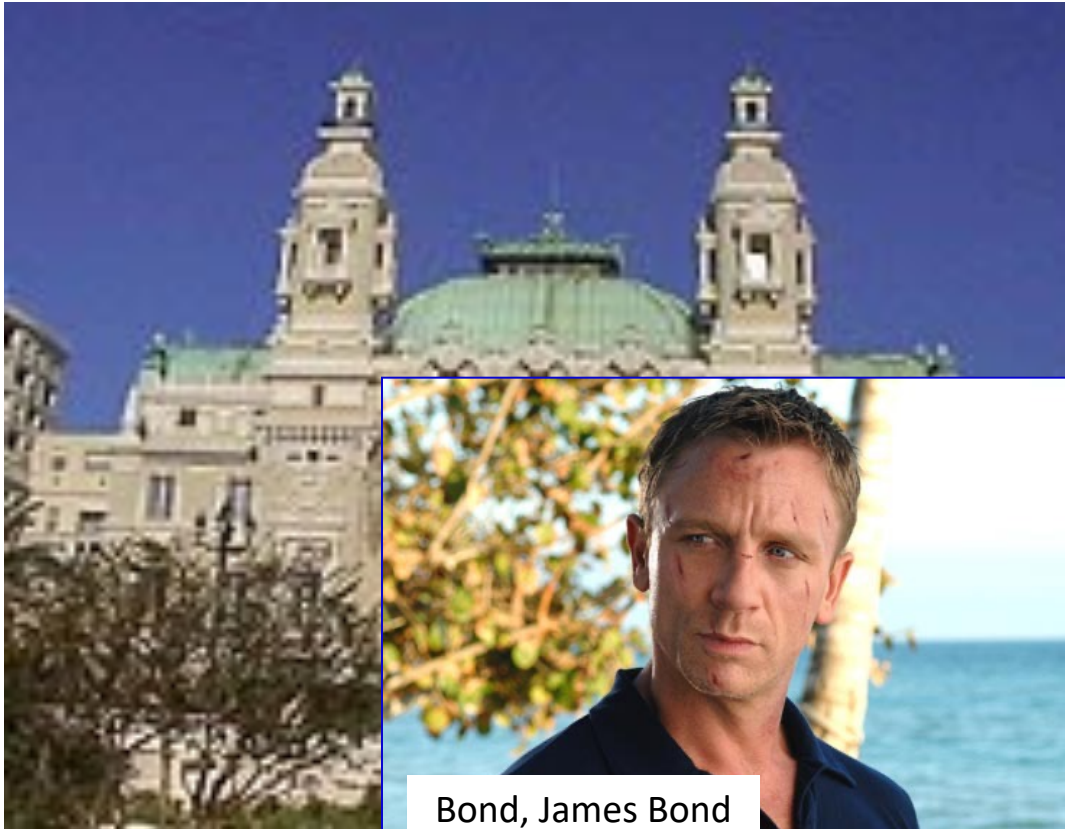


ights via
om trials

te Carlo
ds, Math/CS

The two *Monte Carlos*

and their denizens...



Bond, James Bond

Monte Carlo casino, Monaco



Ulam, Stan Ulam

Monte Carlo
methods, Math/cs

Nature ~ Monte Carlo process

Universe: "Hah! I'm too random to be tamed!"

Nature: I see. Um – what about this very plain 'rock'?
No, the one over here..."

Universe: "On it! Hah! Now it's a mess!!!"

Nature: "Many thanks ~ You do you!"

Nature ~ Monte Carlo process

Universe: "Hah! I'm too random to be tamed!"

Nature: I see. Um – what about this very plain 'rock'?
No, the one over here..."

Universe: "On it! Hah! Now it's a mess!!!"

Nature: "Many thanks ~ You do you!"

Nature ~ Monte Carlo process

Universe: "Hah! I'm too random to be tamed!"

Nature: I see. Um – what about this very plain 'rock'?
No, the one over here..."

Universe: "On it! Hah! Now it's a mess!!!"

Nature: "Many thanks ~ You do you!"

A "Monte Carlo" function...

```
from random import *
```

```
def guess( hidden ):
```

```
    """ tries to guess our hidden number  
    """
```

Remember, this is the list [0,1,...,98,99]

```
    compguess = choice( list(range(100)) )
```

```
    if compguess == hidden:
```

```
        return 1
```

```
    else:
```

```
        return 1 + guess( hidden )
```



one guess here -- plus -- all "future" guesses!



Monte Carlo *dice*

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?

doubles check?

one doubles here, plus all the "future" rolls!

Monte Carlo *Curtains*

Monte Carlo *Curtains*

Let's Make
a Deal

'09-now



1/29/2024

S15 E83 Jan 29, 2024

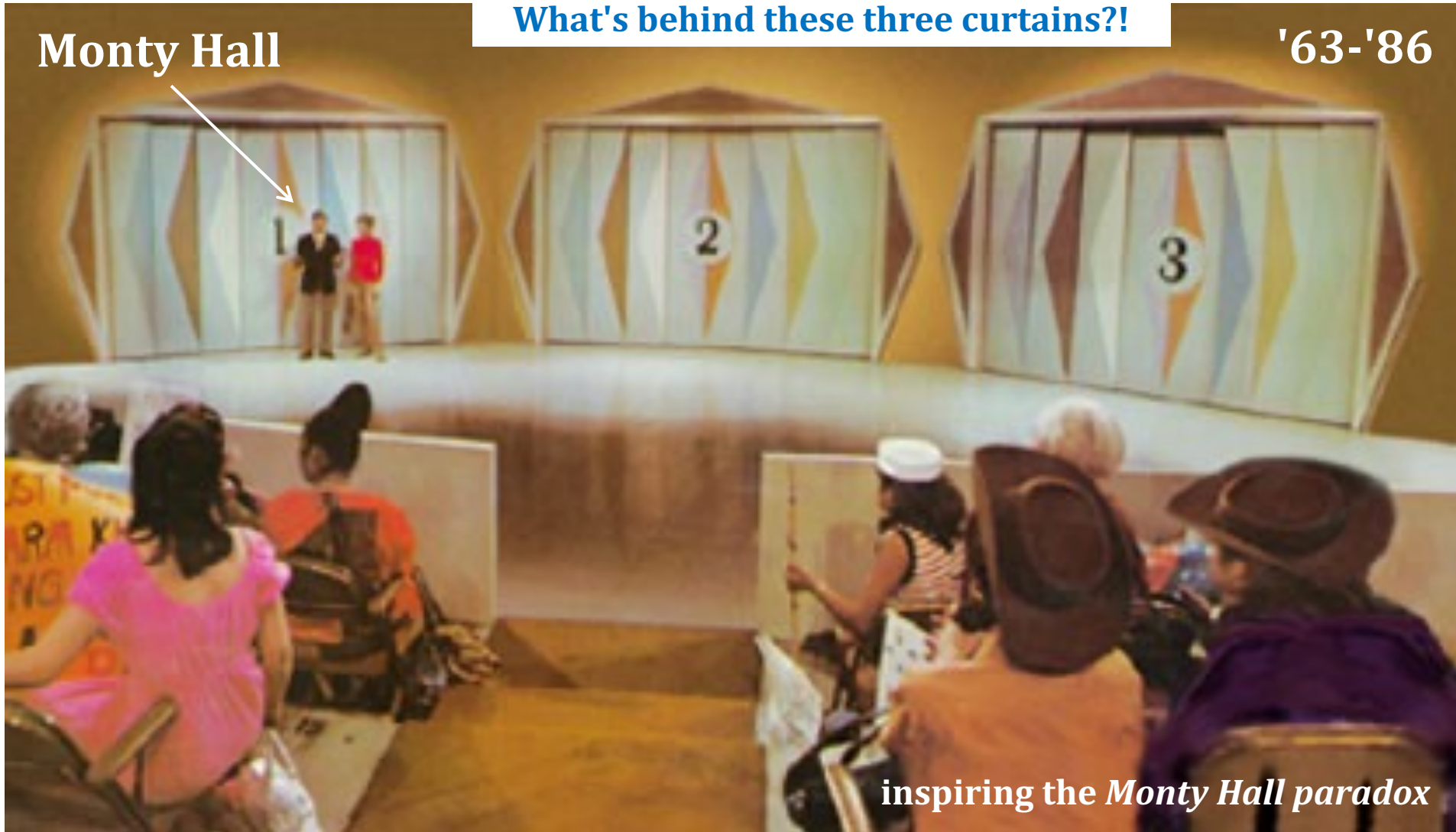
This LMAD cast an entertaining spell!! A cyclist tries to ride her way into a \$10,000 "pair-a-dice", Then, a hot dog plays "car pong" for a hot ride. Plus, Wayne & Jonathan engage in a prize "debate" & a dino rider gets a "fortune cookie" and jams with Cat. Wayne Brady hosts.

Monte Carlo *Curtains*

Monty Hall

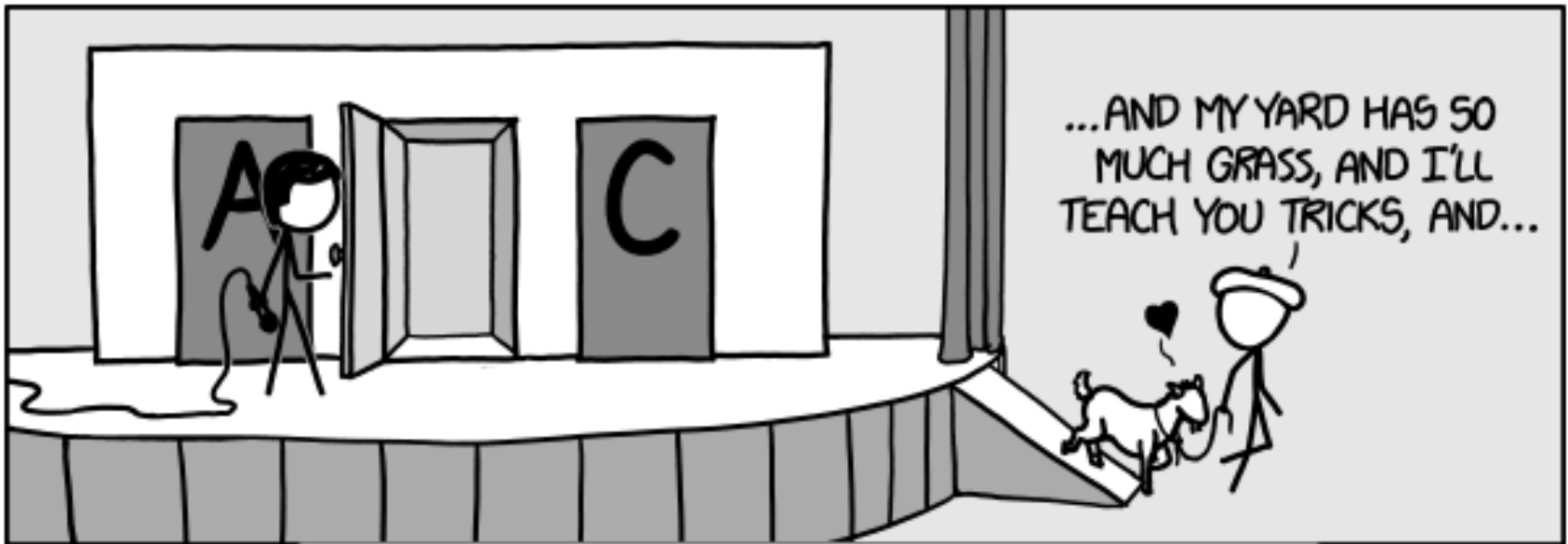
What's behind these three curtains?!

'63-'86



XKCD's take...

MONTY HALL



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

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

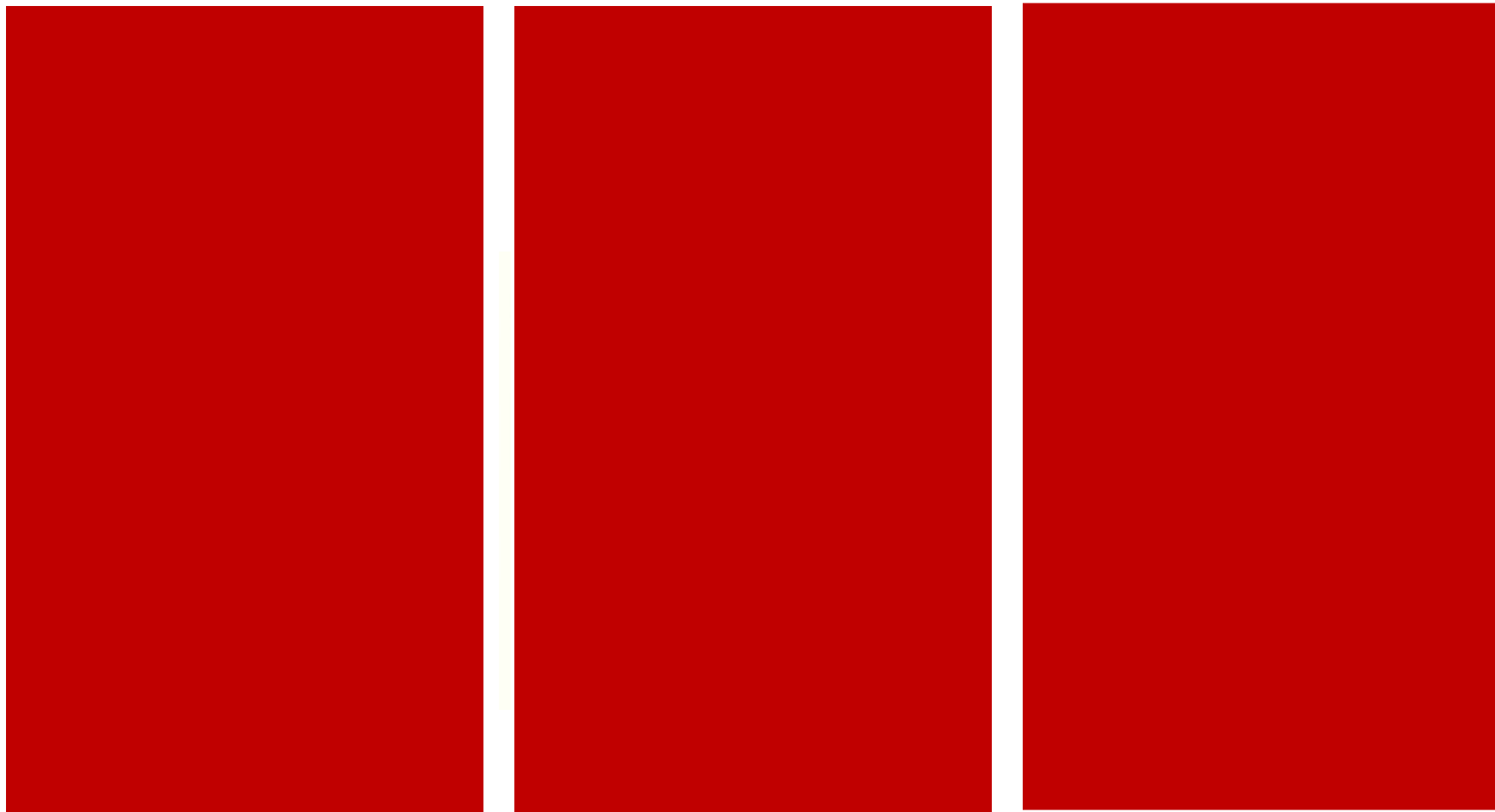
Monte Carlo Monty Hall

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



Let's play!

Monte Carlo Monty Hall

Annotations:

- Your initial choice! (points to `init`)
- 'switch' or 'stay' (points to `sors`)
- number of times to play (points to `N`)

```
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, ':', result)

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



0

1

2

3

4

5

6

7

8

9

A

B

C

D

E

F

G

H



[illegible]

What if you win some spam?!

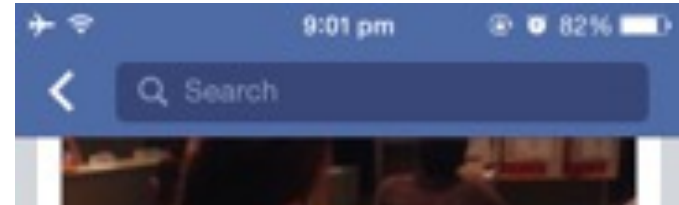
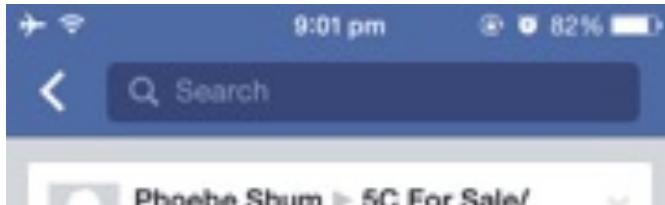
What if you win some spam?!



What if you win some spam?!



What if you win some spam?!



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



Write a comment...



Post



News Feed



Requests



Messenger



Notifications



More



Rosh Lam

Is this actually for sale? I'm interested in buying, \$1

7 minutes ago · Like · Reply



Write a comment...



Post



News Feed



Requests



Messenger



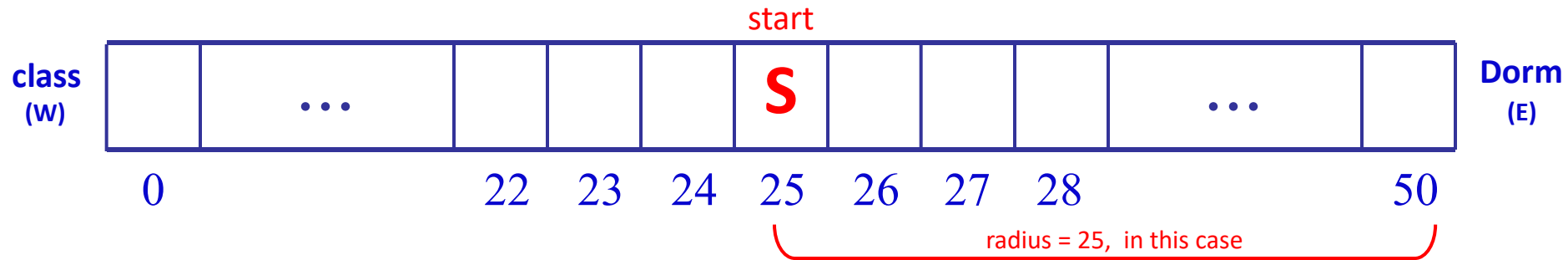
Notifications



More

An example *closer to home*

hw2pr2



Our very-tired student (**S**) leaves H/S after a "late-night" breakfast. Each step, they randomly go toward class (**West**) or the dorms (**East**)

Once the student arrives at the dorm or classroom, the trip is complete.

The program should then return the total number of steps taken.

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

rwpos(st, nsteps)

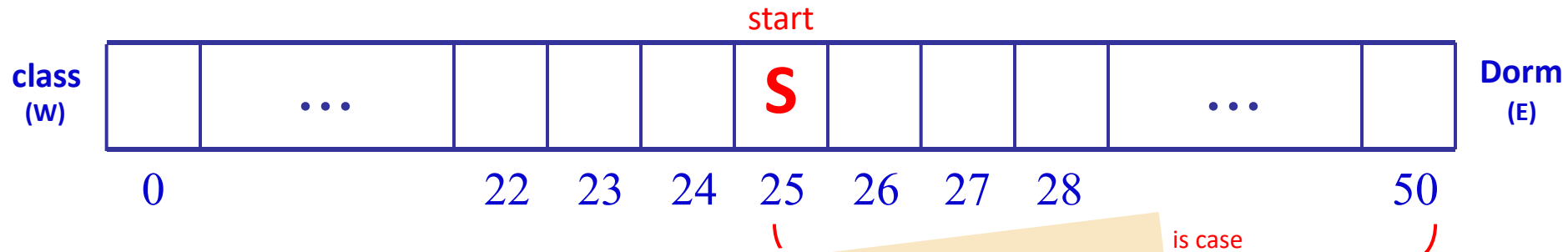
take **nsteps** random
steps starting at **st**

rwsteps(st, low, hi)

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

An example *closer to home*

hw2pr2



Your tasks:

(a) Create a Monte Carlo "ASCII" animation of our random walker...

(b) To build a creative variation of this process with two "walkers" ...

nario...

rwsteps(st, low, hi)

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

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

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



`turtle` — Turtle graphics

Source code: [Lib/turtle.py](https://lib.python.org/2.7/turtle.py)

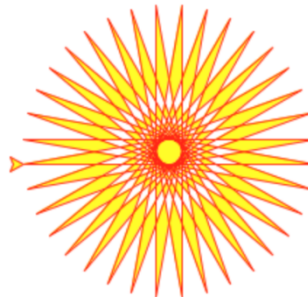
Introduction

Turtle graphics is a popular way for introducing programming to kids. It was part of the original Logo programming language developed by Wally Feurzeig, Seymour Papert and Cynthia Solomon in 1967.

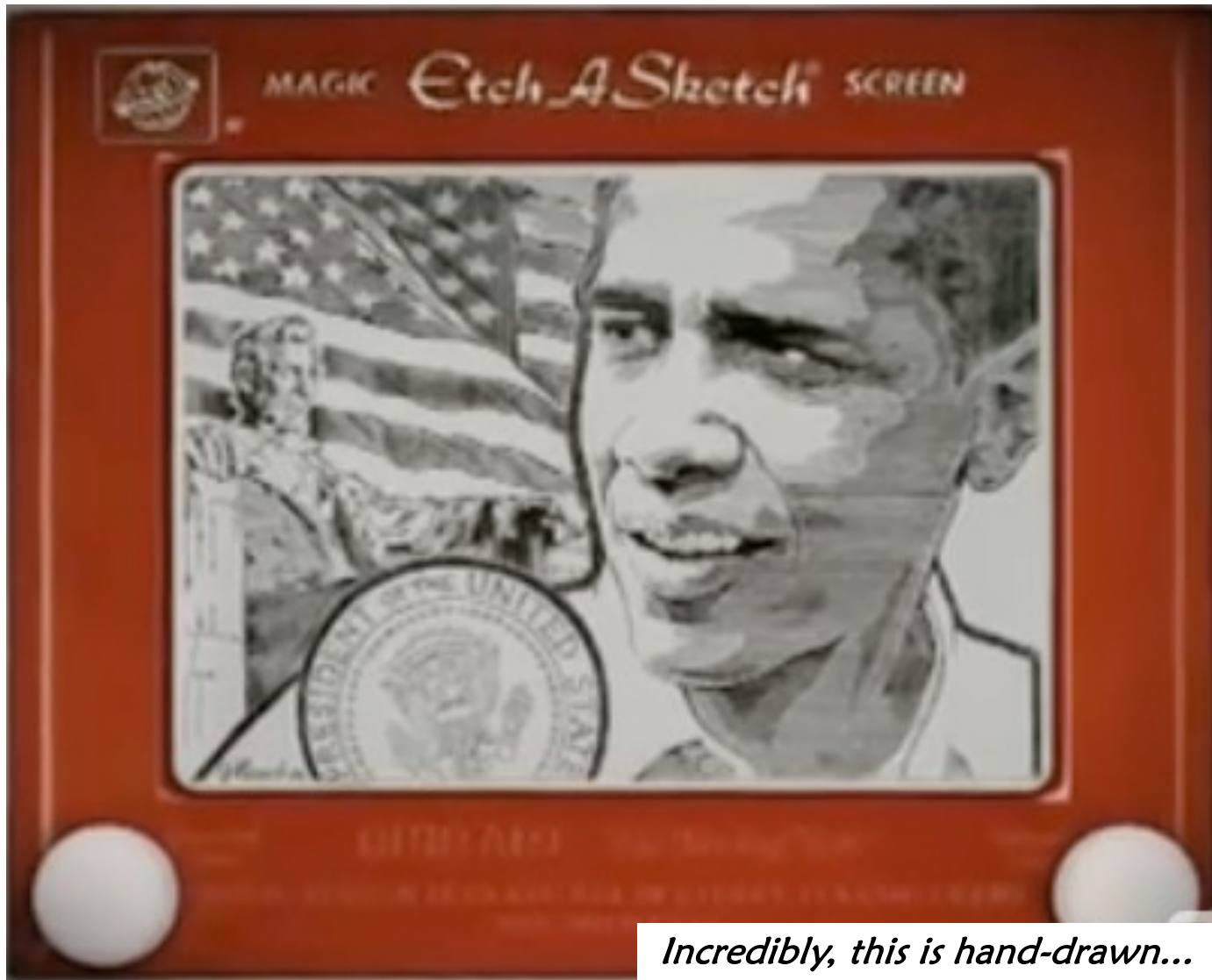
Imagine a robotic turtle starting at (0, 0) in the x-y plane. After an `import turtle`, give it the command `turtle.forward(15)`, and it moves (on-screen!) 15 pixels in the direction it is facing, drawing a line as it moves. Give it the command `turtle.right(25)`, and it rotates in-place 25 degrees clockwise.

Turtle star

Turtle can draw intricate shapes using programs that repeat simple moves.



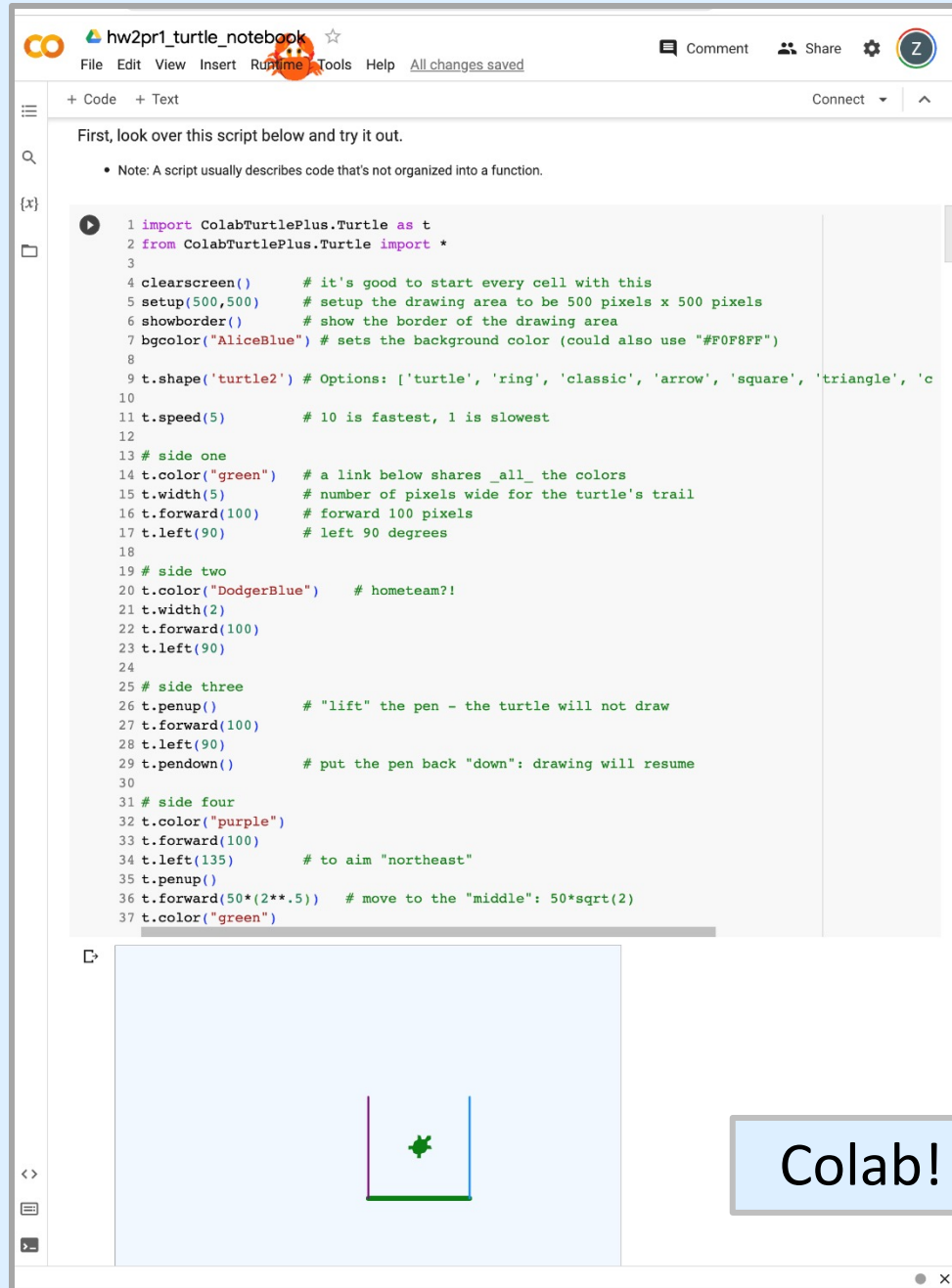
Lab 2 ~ *Python's Etch-a-Sketch*



more *typical* etch-a-sketch result



In-browser Python...



hw2pr1_turtle_notebook

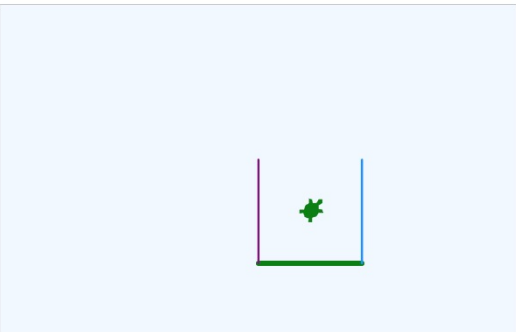
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text Connect

First, look over this script below and try it out.

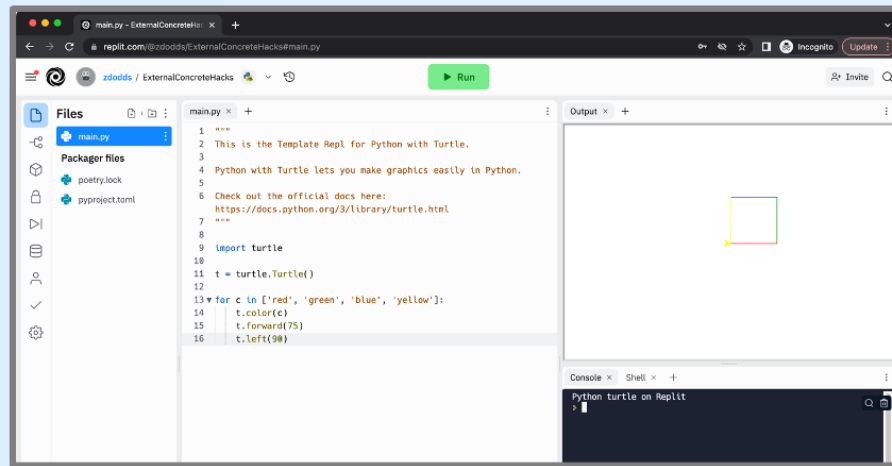
- Note: A script usually describes code that's not organized into a function.

```
1 import ColabTurtlePlus.Turtle as t
2 from ColabTurtlePlus.Turtle import *
3
4 clearscreen() # it's good to start every cell with this
5 setup(500,500) # setup the drawing area to be 500 pixels x 500 pixels
6 showborder() # show the border of the drawing area
7 bgcolor("AliceBlue") # sets the background color (could also use "#F0F8FF")
8
9 t.shape('turtle2') # Options: ['turtle', 'ring', 'classic', 'arrow', 'square', 'triangle', 'c
10
11 t.speed(5) # 10 is fastest, 1 is slowest
12
13 # side one
14 t.color("green") # a link below shares _all_ the colors
15 t.width(5) # number of pixels wide for the turtle's trail
16 t.forward(100) # forward 100 pixels
17 t.left(90) # left 90 degrees
18
19 # side two
20 t.color("DodgerBlue") # hometeam?!
21 t.width(2)
22 t.forward(100)
23 t.left(90)
24
25 # side three
26 t.penup() # "lift" the pen - the turtle will not draw
27 t.forward(100)
28 t.left(90)
29 t.pendown() # put the pen back "down": drawing will resume
30
31 # side four
32 t.color("purple")
33 t.forward(100)
34 t.left(135) # to aim "northeast"
35 t.penup()
36 t.forward(50*(2**.5)) # move to the "middle": 50*sqrt(2)
37 t.color("green")
```

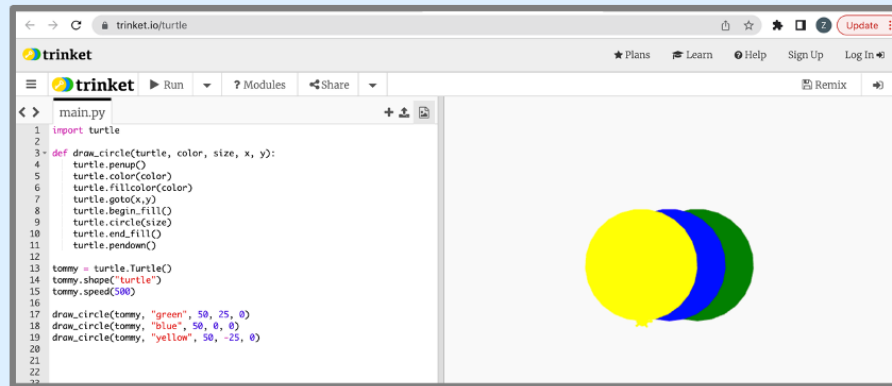


Colab!

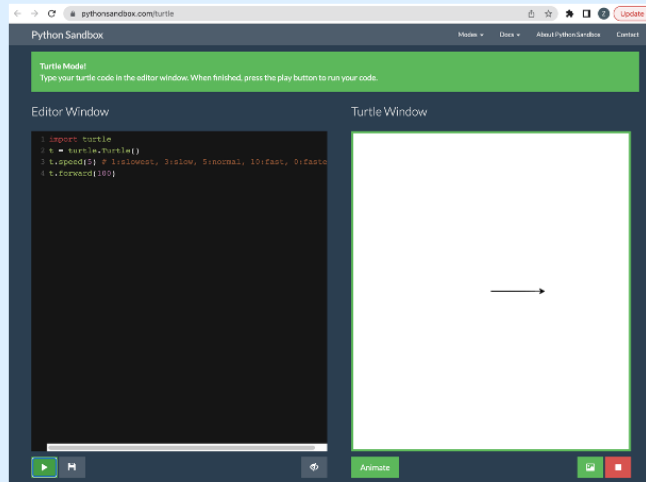
In-browser alternatives...



repl.it



Trinket



Python
sandbox

Single-path recursion

A starter *script*:

```
# a triangle
# as a _script_
forward(100)
left(120)
forward(100)
left(120)
forward(100)
left(120)
```



a *script* is code that runs on the "left margin" of a Python file (aka, the "west coast")

And a starter *function*:

```
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
```

tri(3)

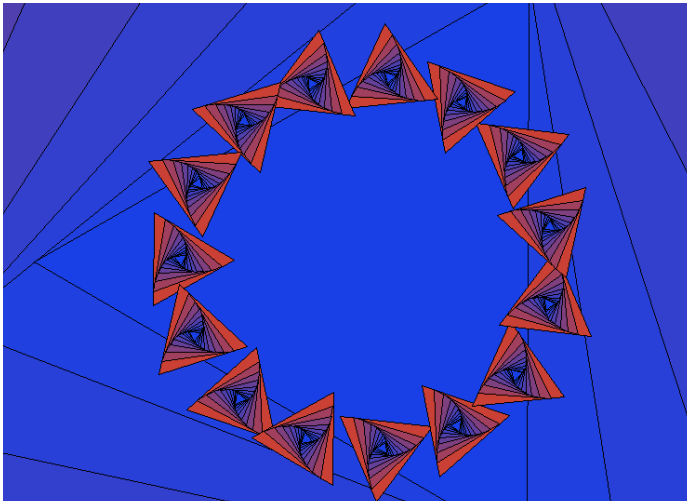


I don't know about `tri`, but there's *no* `return` ... !



Turtle's ability?

It varies...

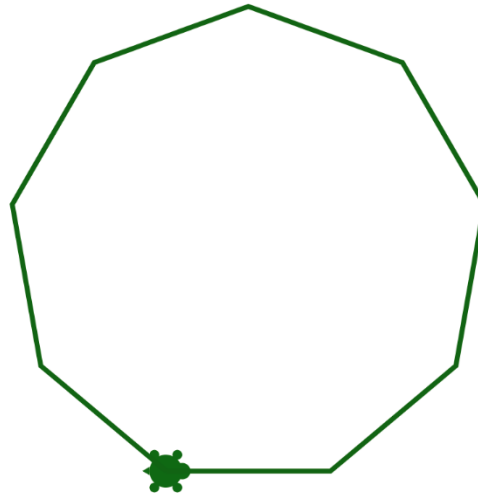


it can vary *widely*

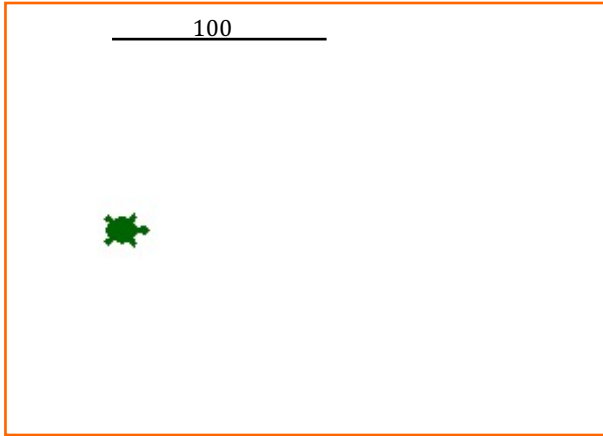
```
def poly(n,N):  
    """ n == sides to go (to be drawn) [[varies]]  
        N == total # of sides in the regular polygon [[constant]]  
    """  
    if n == 0:  
        return # stop!  
    else:  
        # print("side", n)  
        t.forward(100)  
        angle = 360/N  
        t.left(angle)  
        poly(n-1,N) # draw the remaining sides...
```

poly(9,9)

Help! Grid On/Off



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



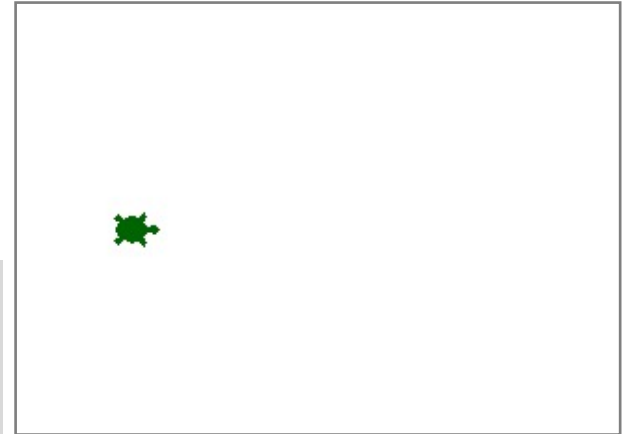
```
def chai(dist):  
    """ mystery fn! """  
    if dist < 20:  
        return  
    else:  
        forward(dist)  
        left(90)  
        forward(dist/2.0)  
        right(90)  
        # line (a)  
        right(90)  
        forward(dist)  
        left(90)  
        # line (b)  
        left(90)  
        forward(dist/2.0)  
        right(90)  
        backward(dist)
```

Be the turtle !

(2a)

Imagine replacing
line (a)
with the line
chai(dist/2)

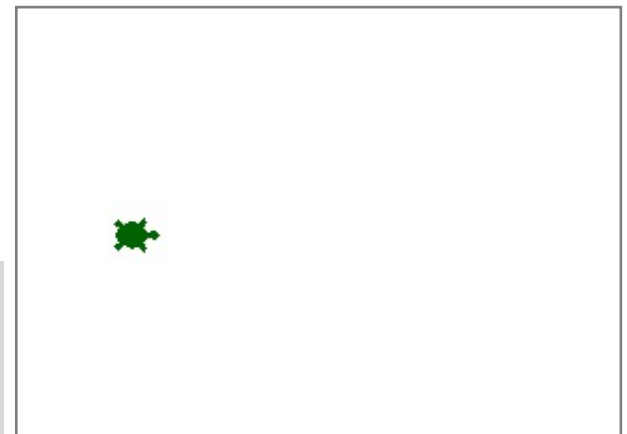
What would
chai(100) then
draw?



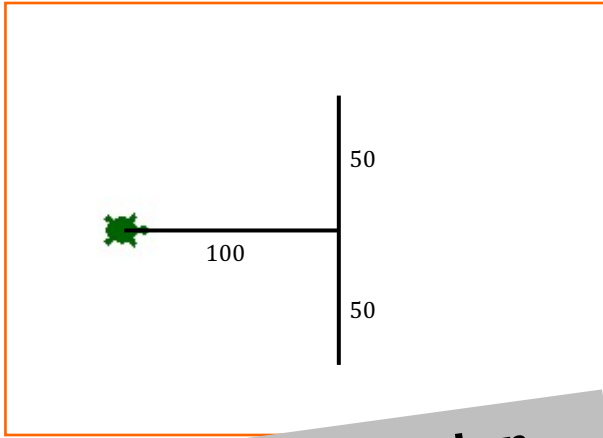
(2b)

then, ALSO replace
line (b)
with the same line:
chai(dist/2)

What would
chai(100) then
draw?!



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



no recursion

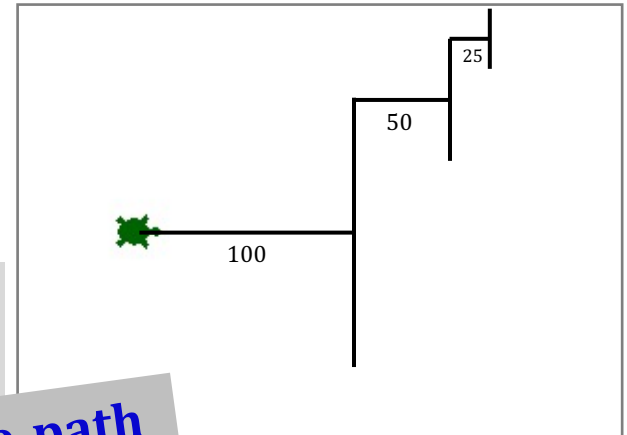
```
def chai(dist):  
    """mystery fn! """  
    if dist < 20:  
        return  
    else:  
        forward(dist)  
        left(90)  
        forward(dist/2.0)  
        right(90)  
        # line (a)  
        right(90)  
        forward(dist)  
        left(90)  
        # line (b)  
        left(90)  
        forward(dist/2.0)  
        right(90)  
        backward(dist)
```

Be the turtle !

(2a)

Imagine replacing
line (a)
with the line
chai(dist/2)

What would
chai(100) then
draw?

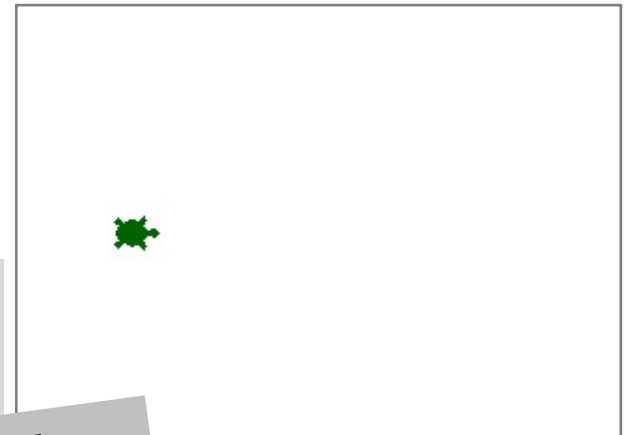


**single-path
recursion**

(2b)

then, ALSO replace
line (b)
with the same line:
chai(dist/2)

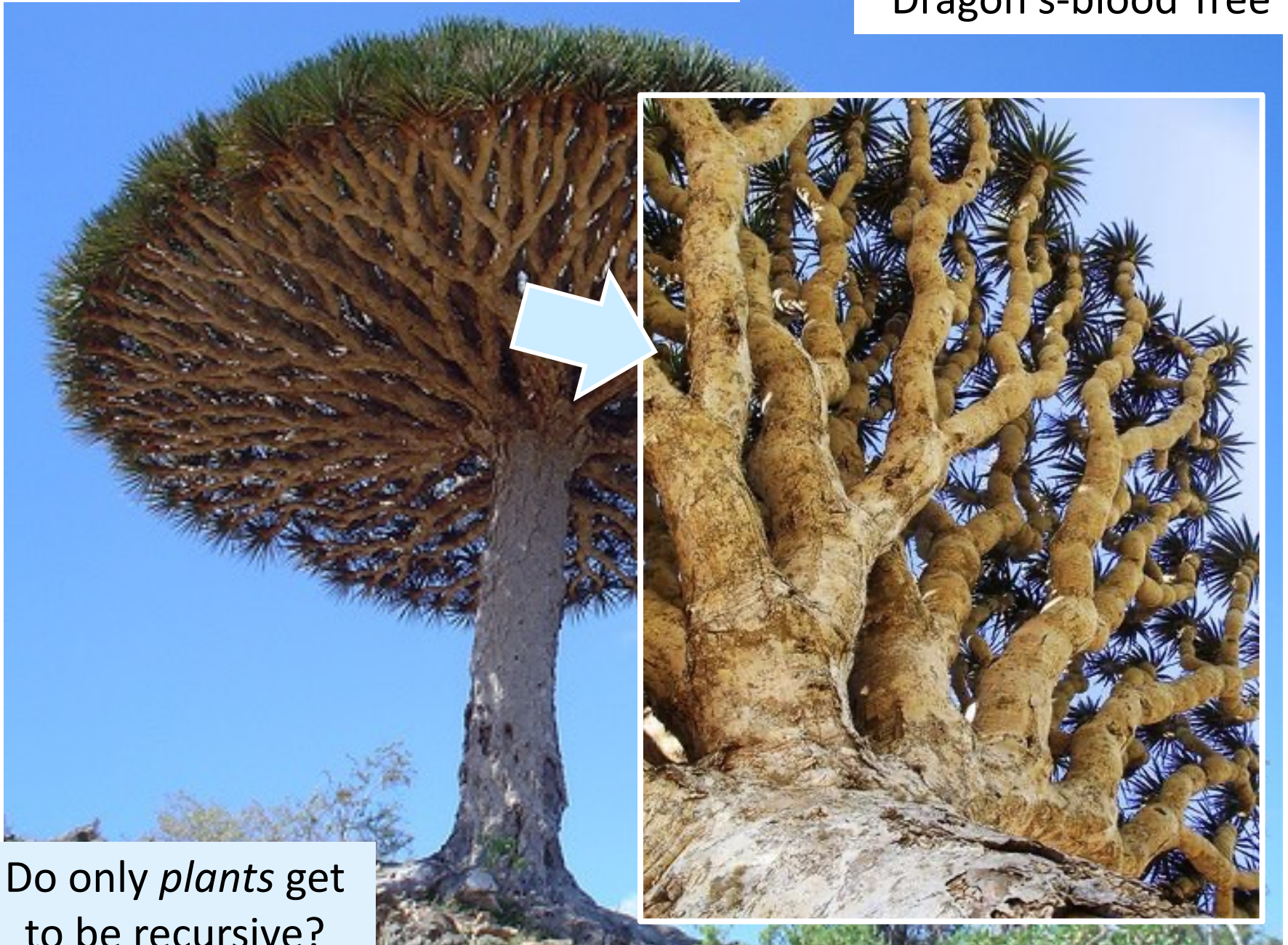
What would
chai(100) then
draw?!



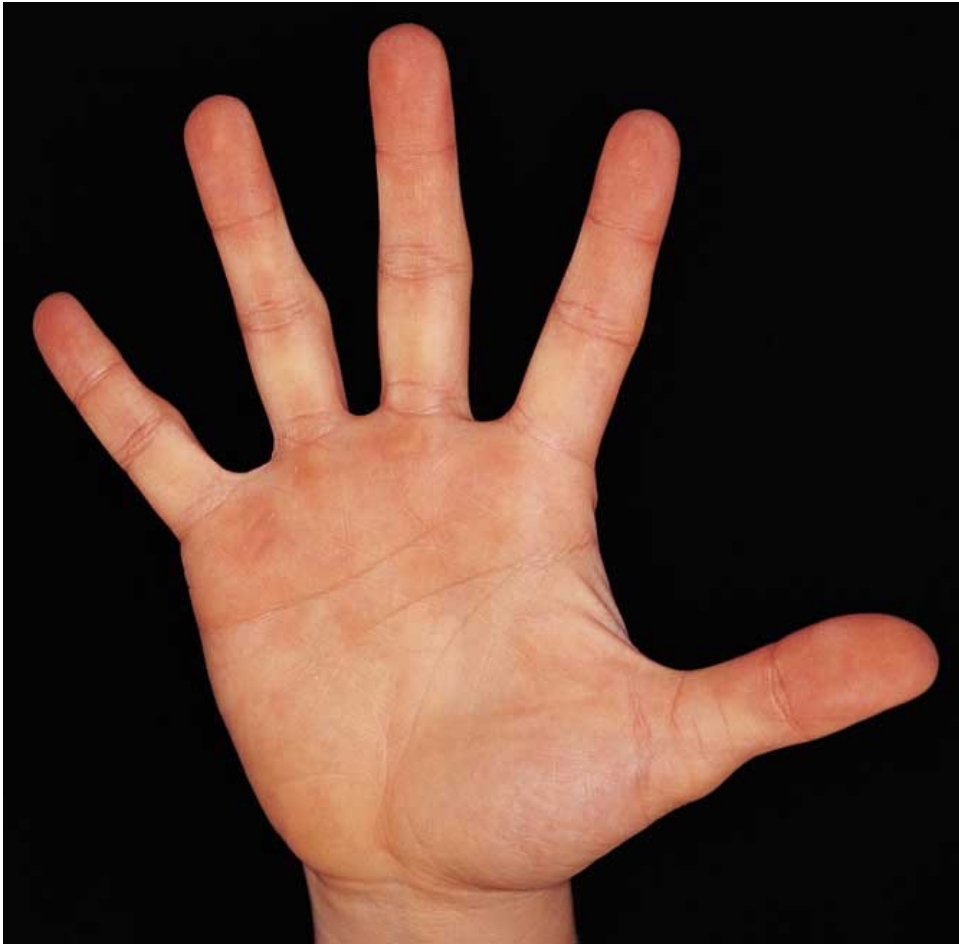
**branching
recursion**

A brief word from our sponsor, Mother Nature...

Dragon's-blood Tree



Do only *plants* get
to be recursive?



Branching seems to be *plants-only*?



Branching recursion is *Strange*!



Br

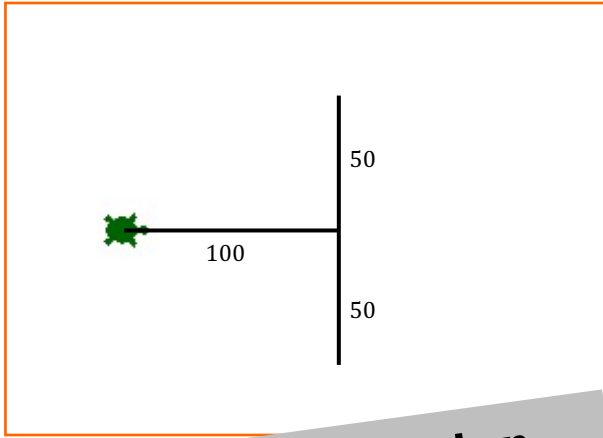
one layer
more!

Cyriak: *conceptually disruptive* recursion...



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

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



no recursion

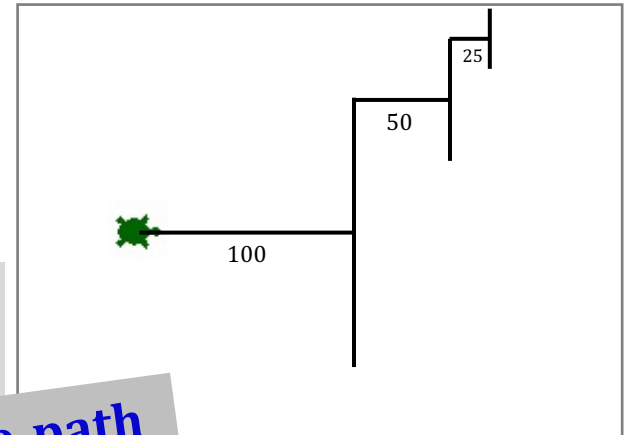
```
def chai(dist):  
    """mystery fn! """  
    if dist < 20:  
        return  
    else:  
        forward(dist)  
        left(90)  
        forward(dist/2.0)  
        right(90)  
        # line (a)  
        right(90)  
        forward(dist)  
        left(90)  
        # line (b)  
        left(90)  
        forward(dist/2.0)  
        right(90)  
        backward(dist)
```

Be the turtle !

(2a)

Imagine replacing
line (a)
with the line
chai(dist/2)

What would
chai(100) then
draw?

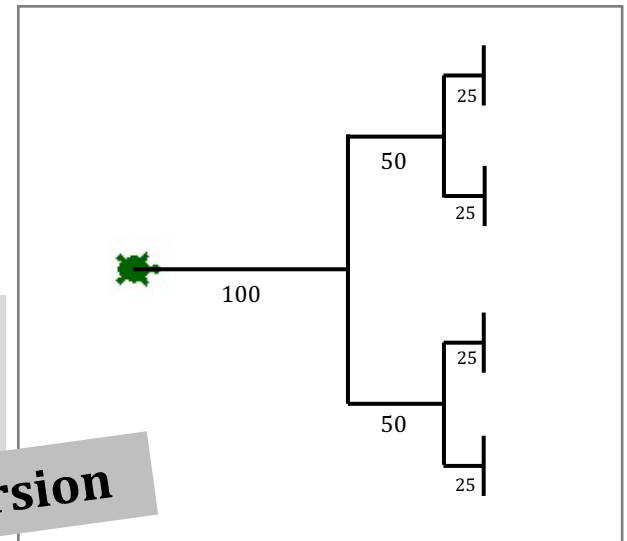


**single-path
recursion**

(2b)

then, ALSO replace
line (b)
with the same line:
chai(dist/2)

What would
chai(100) then
draw?!

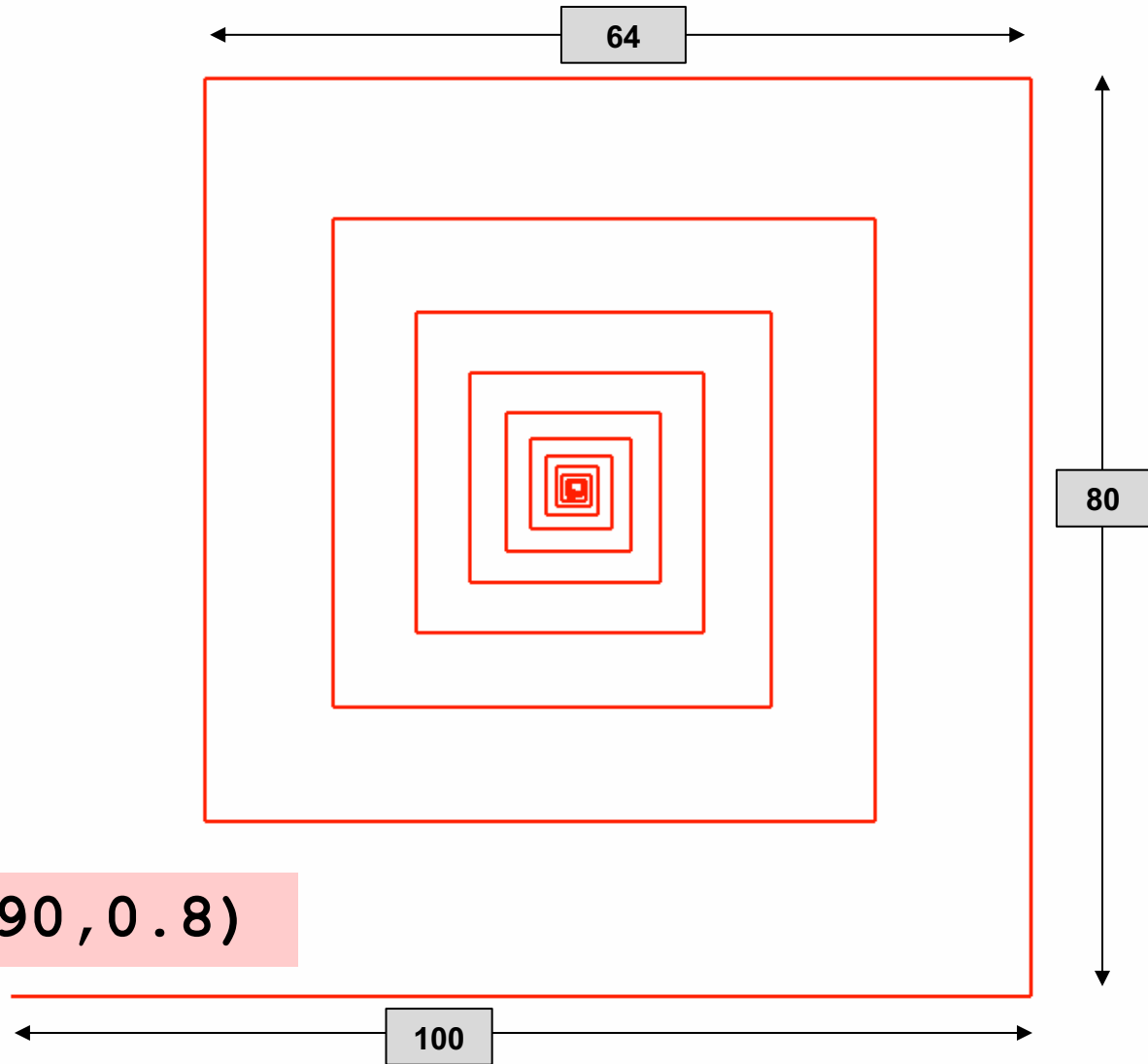


**branching
recursion**

lab ~ hw2pr1

Single-path or *Branching* recursion here?

fractal art



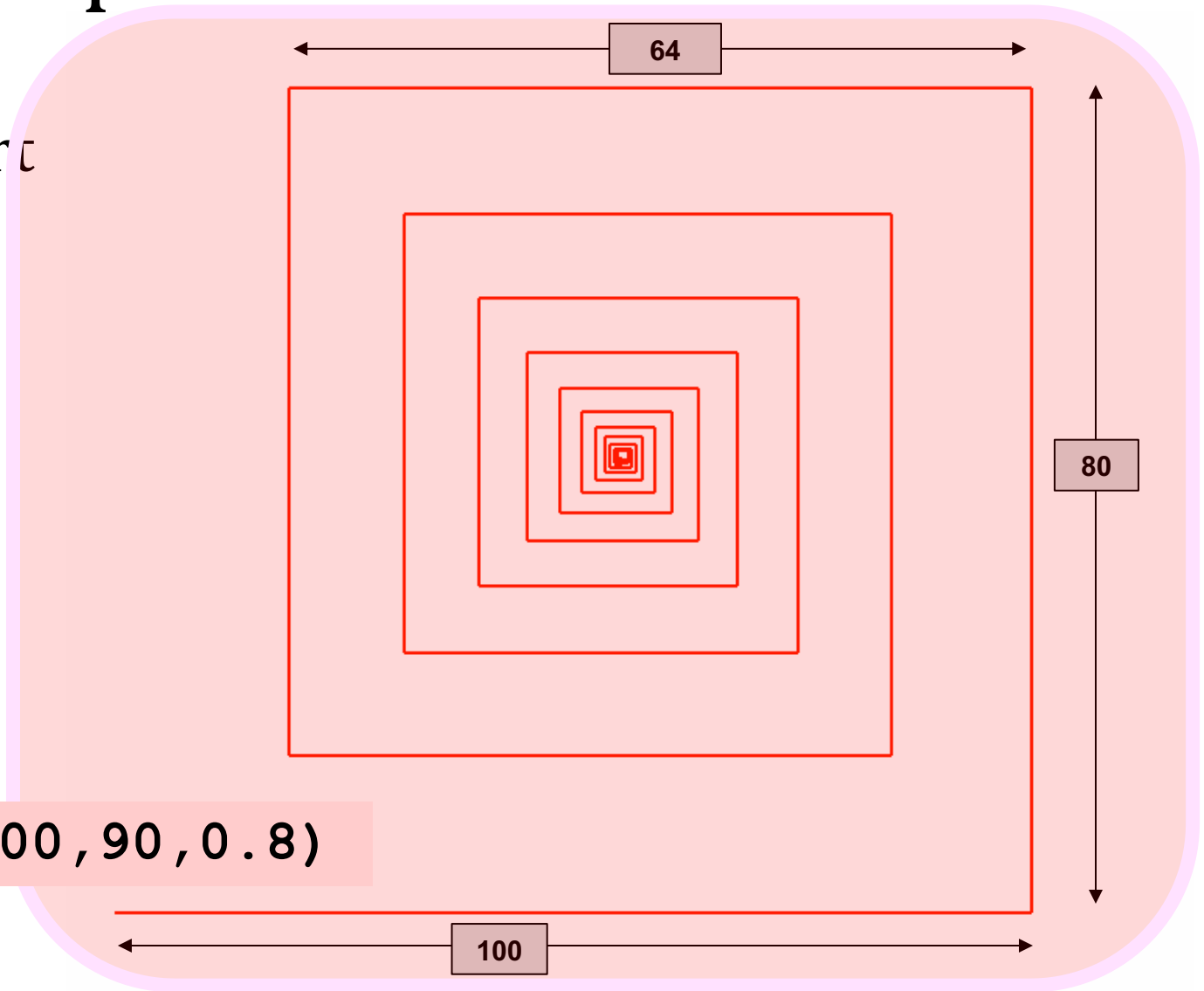
```
spiral(100,90,0.8)
```

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

lab ~ hw2pr1

Single-path!

fractal art



```
spiral(100,90,0.8)
```

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

lab ~ hw2pr1

Single-path!

fractal art

`spiral (80, 90, 0.8)`

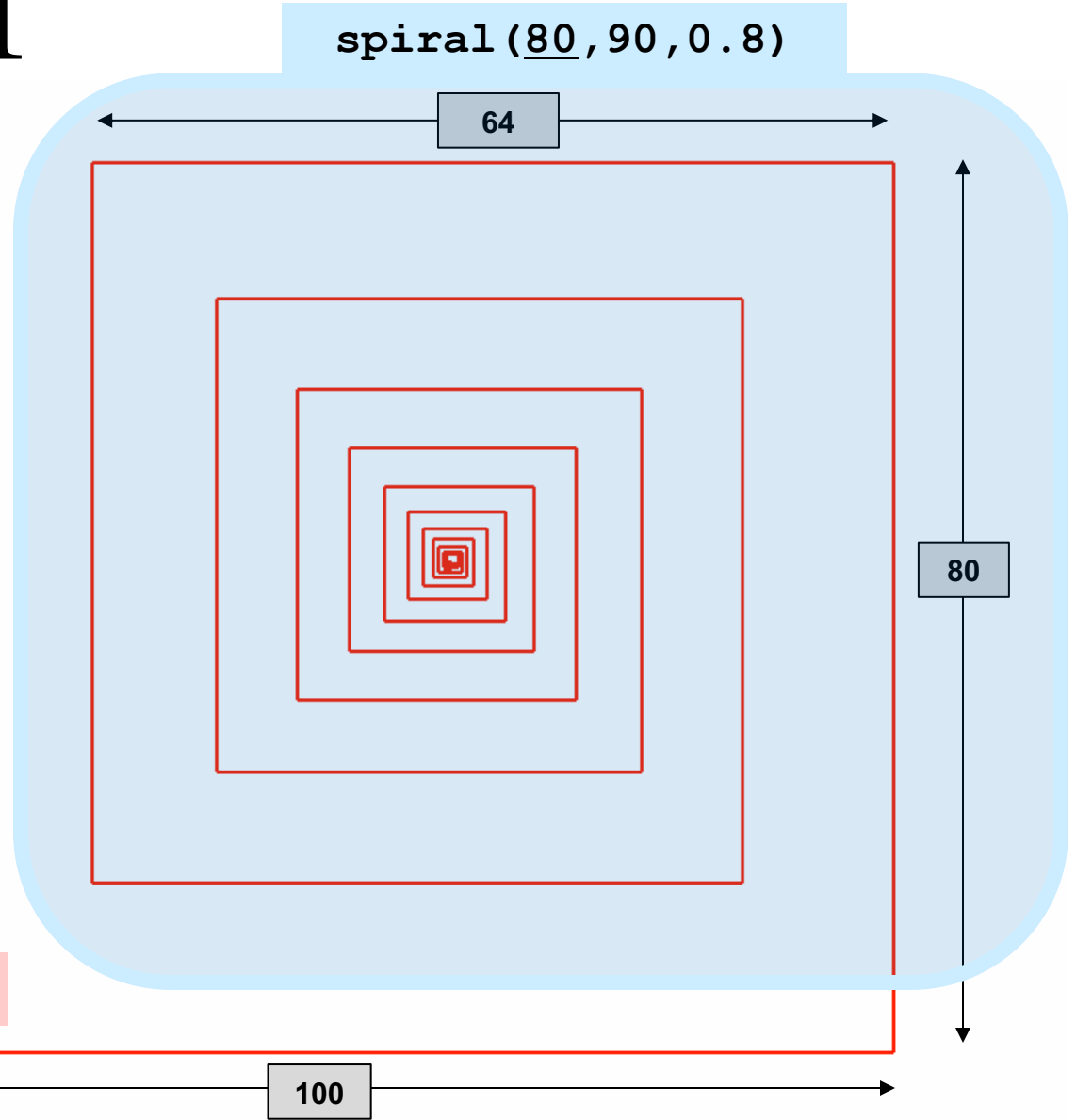
64

80

`spiral (100, 90, 0.8)`

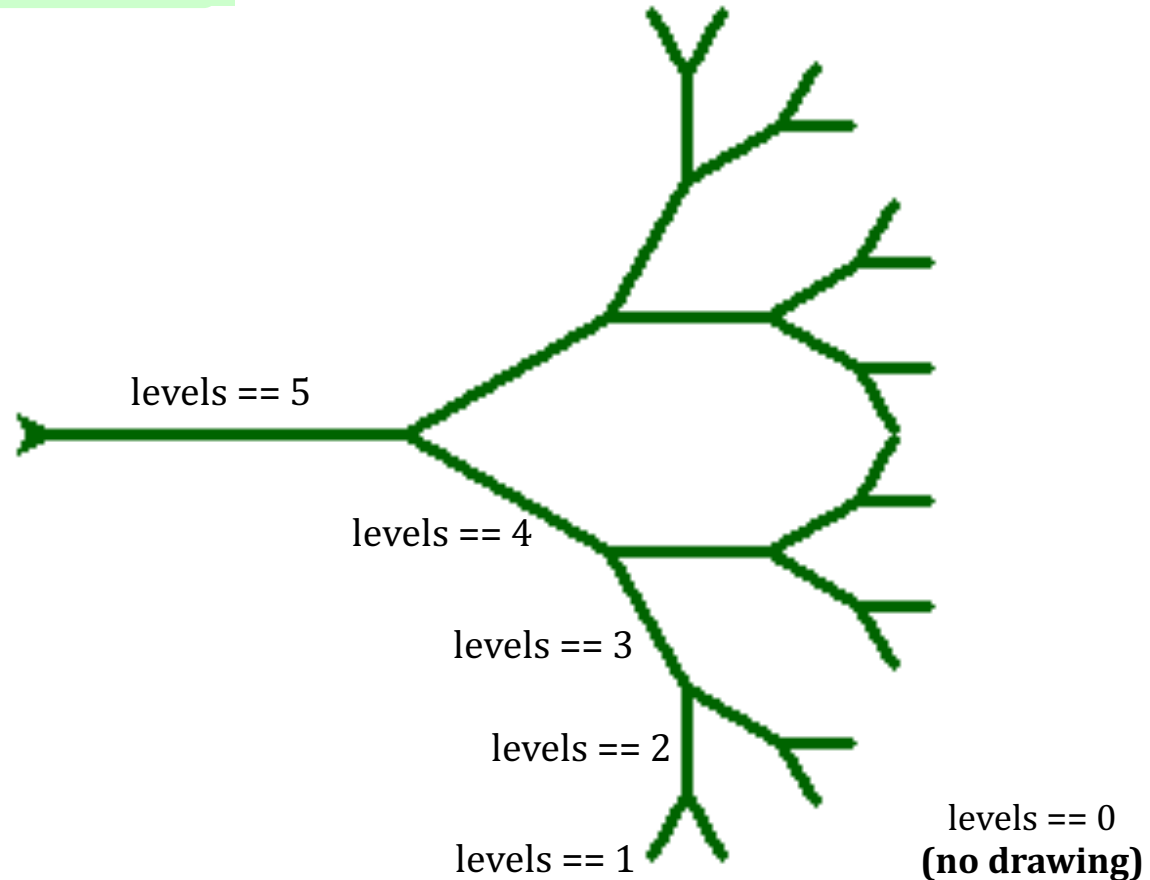
100

`spiral(initLength, angle, multiplier)`



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

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

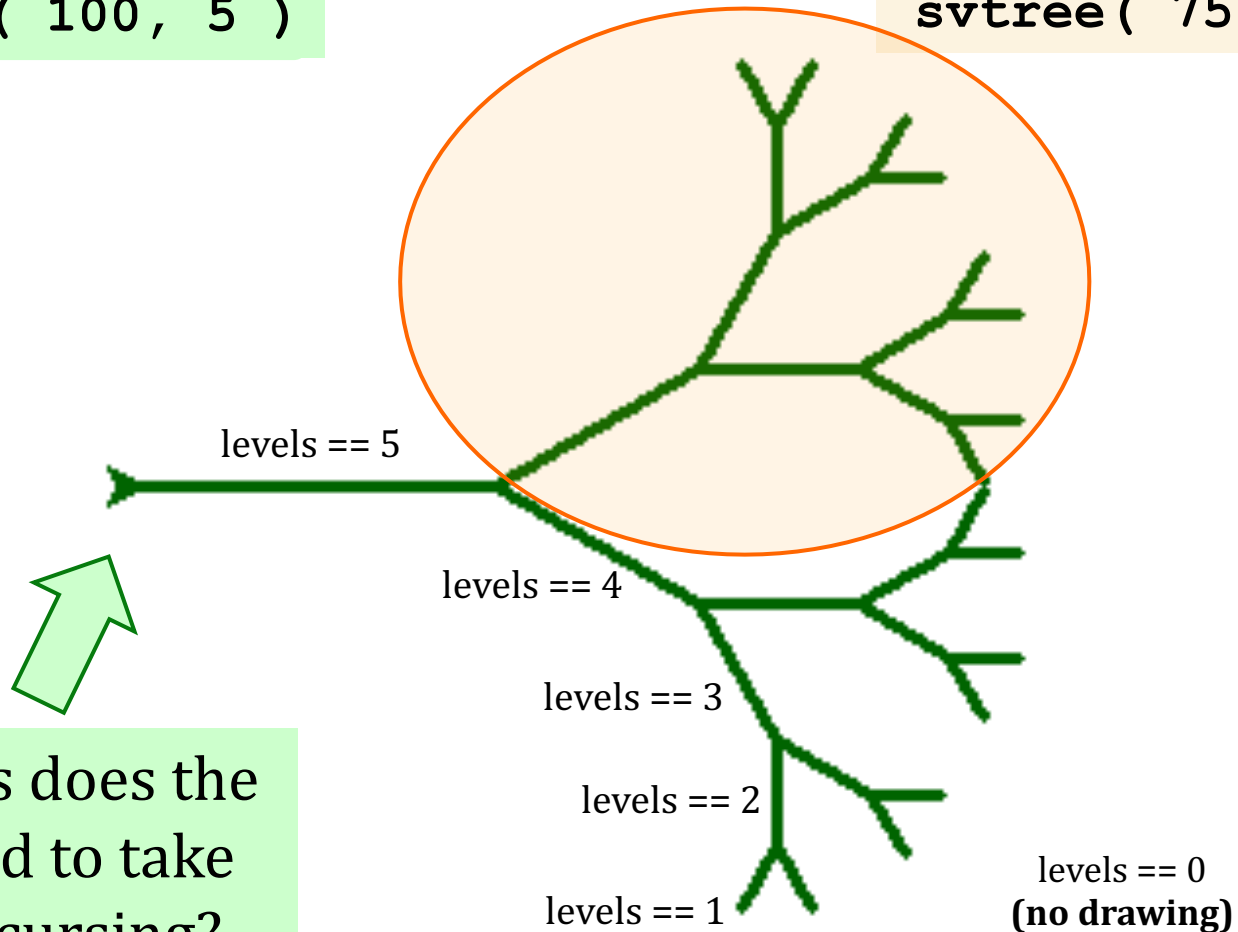


Single-path or *Branching* recursion here?

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

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

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



What steps does the
turtle need to take
before recursing?

Branching recursion!

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

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

step #3: draw a
smaller svtree!

step #2: turn a bit...

step #1: go forward...

levels == 5

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

step #4: turn to
another heading

levels == 4

levels == 3

levels == 2

levels == 1

levels == 0
(no drawing)

step #5: draw another
smaller svtree!

Be sure the turtle
always returns to its
starting position!

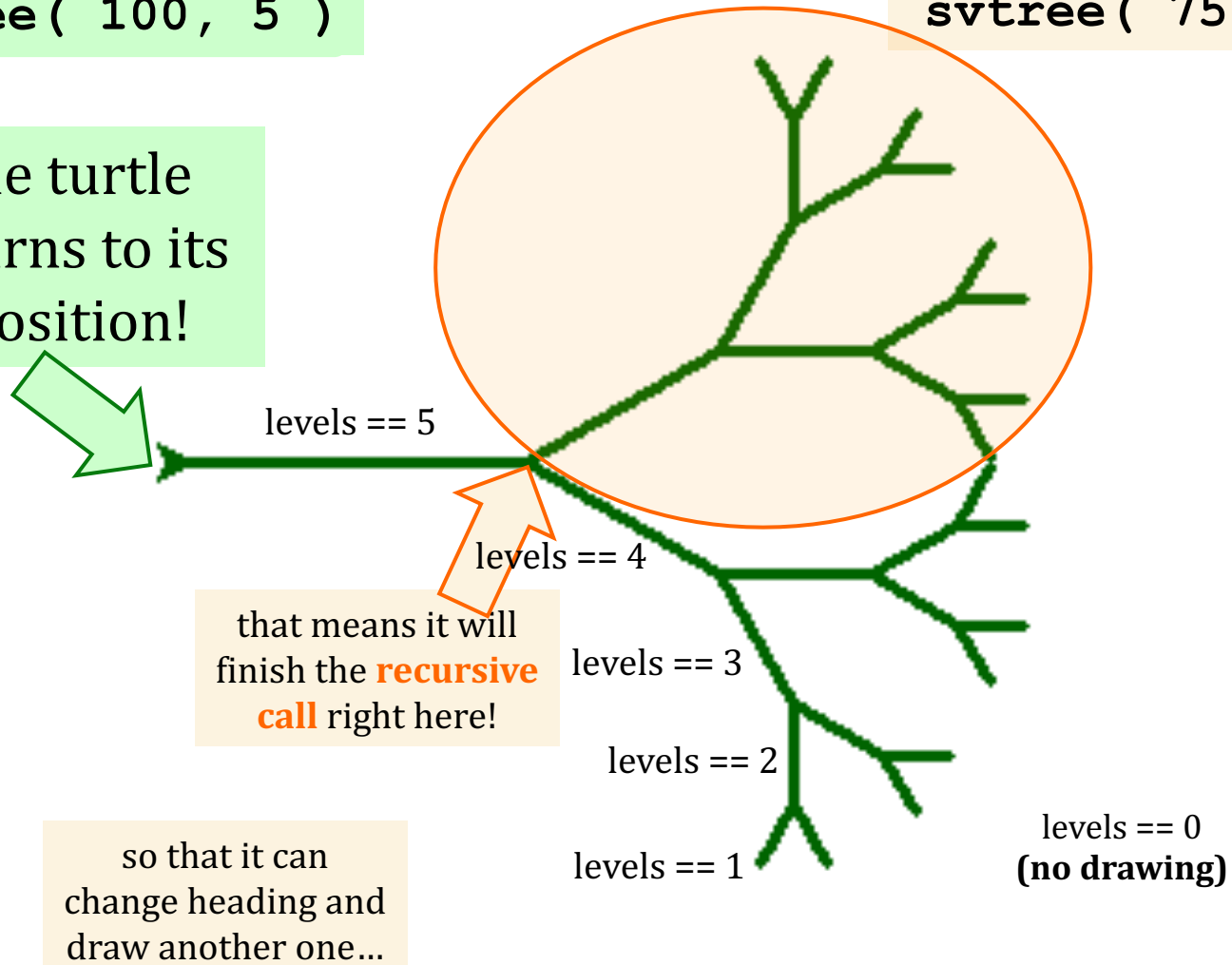
Branching recursion!

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

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

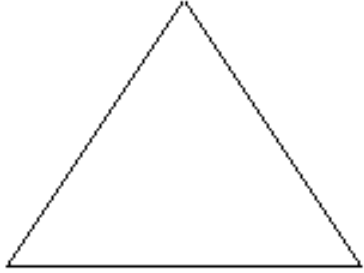
```
svtree( 75, 4 )
```

Be sure the turtle
always returns to its
starting position!

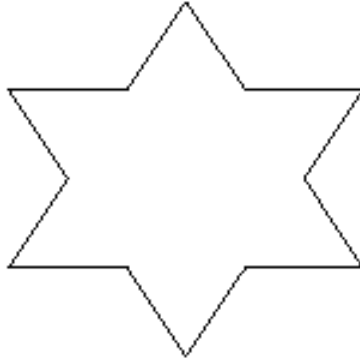


Branching recursion!

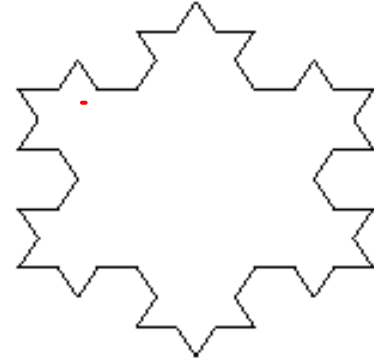
The Koch curve



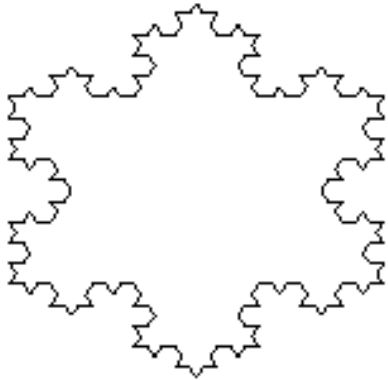
snowflake(100, 0)



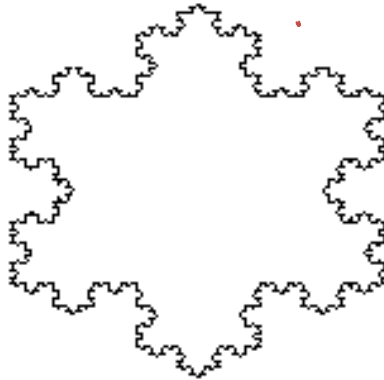
snowflake(100, 1)



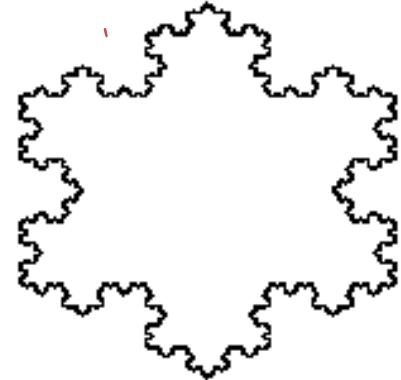
snowflake(100, 2)



snowflake(100, 3)



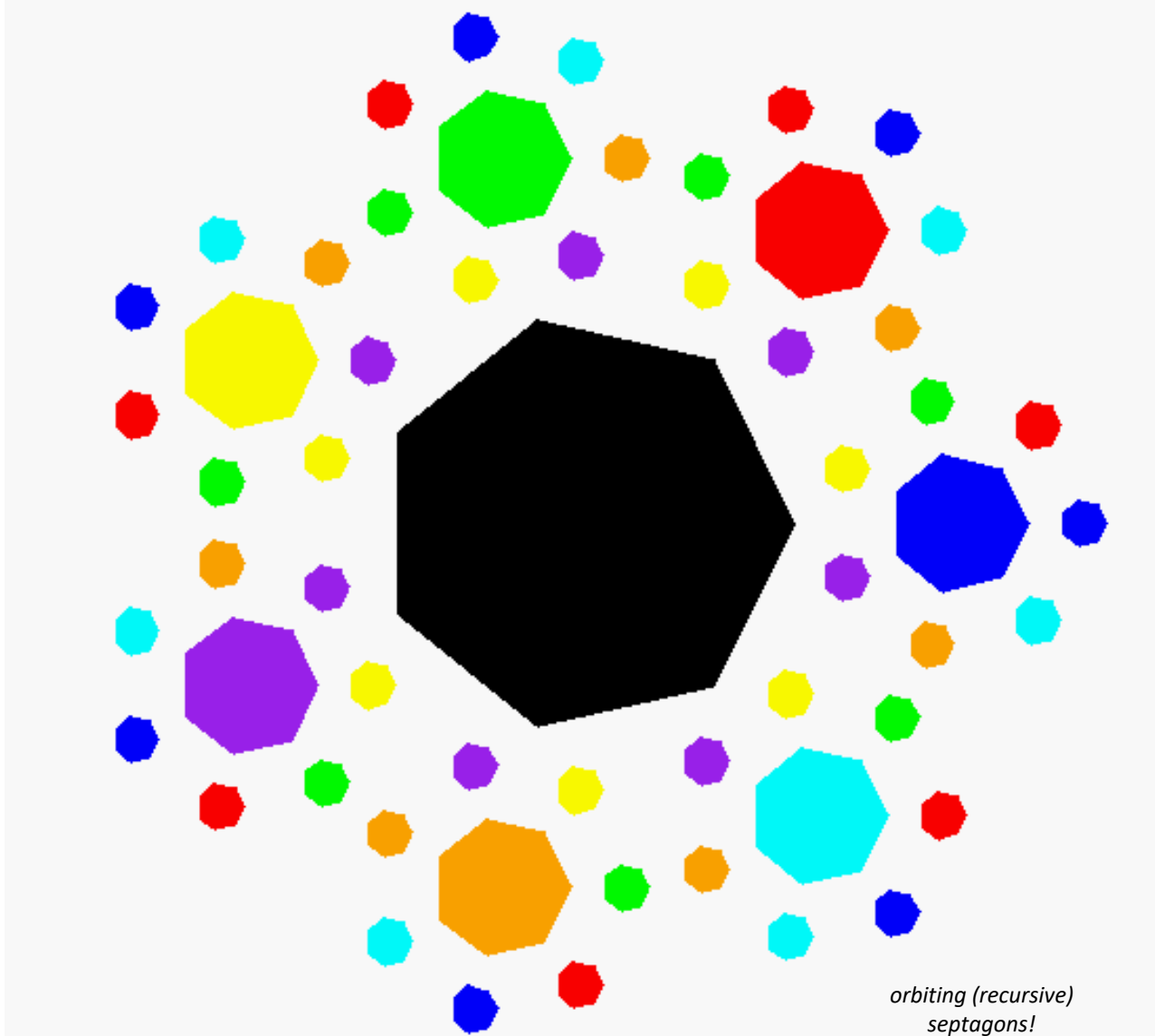
snowflake(100, 4)



snowflake(100, 5)

Recursive turtling!

See you in lab!



*orbiting (recursive)
septagons!*