CS 5 alien on strike!

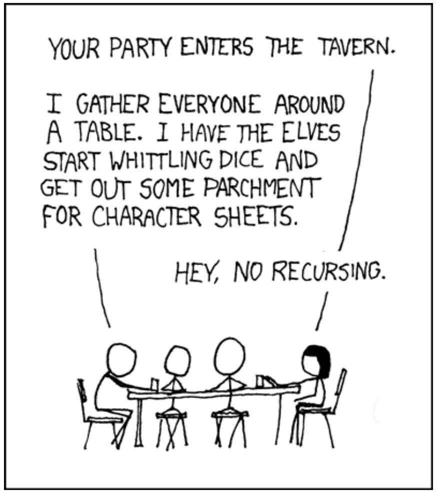
CS 5 green mascot representing today's terrestrial themes

CS 5 Lecture 4

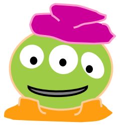
Randomness and More Recursion

Recursion's advantage:

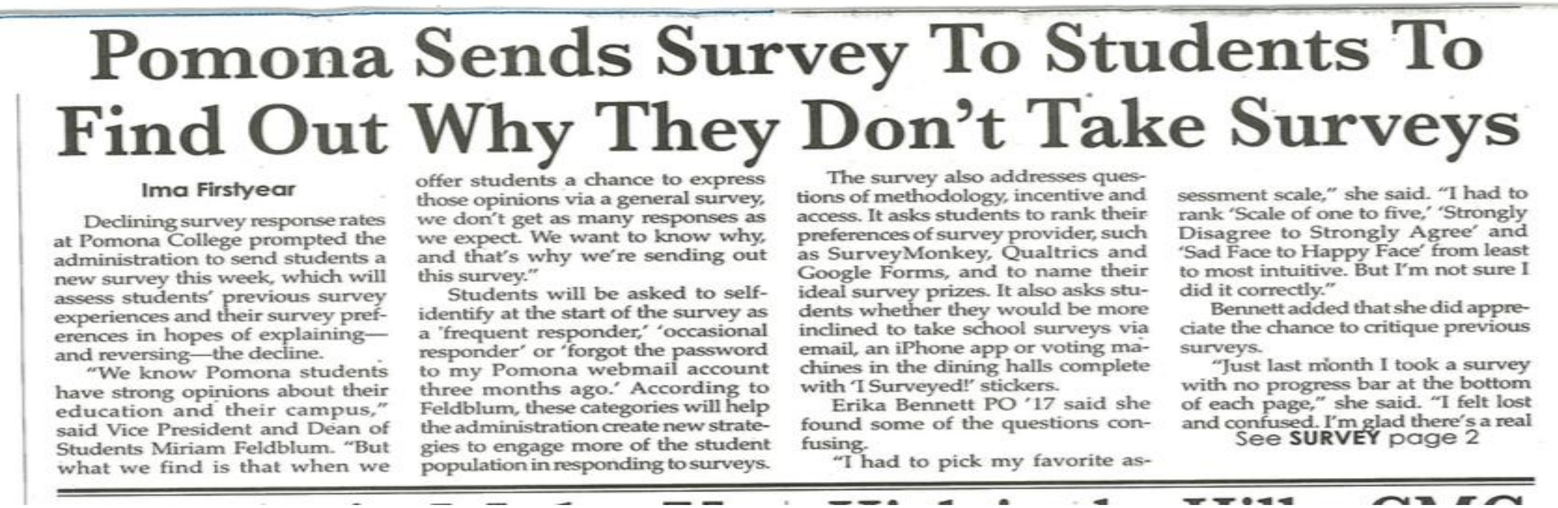
It handles arbitrary structural depth – ***all at once + on its own***!


YOUR PARTY ENTERS THE TAVERN.
I GATHER EVERYONE AROUND A TABLE. I HAVE THE ELVES START WHITTLING DICE AND GET OUT SOME PARCHMENT FOR CHARACTER SHEETS.
HEY, NO RECURSING.

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



As a hat, I'm recursive, too!



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 assessment 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.

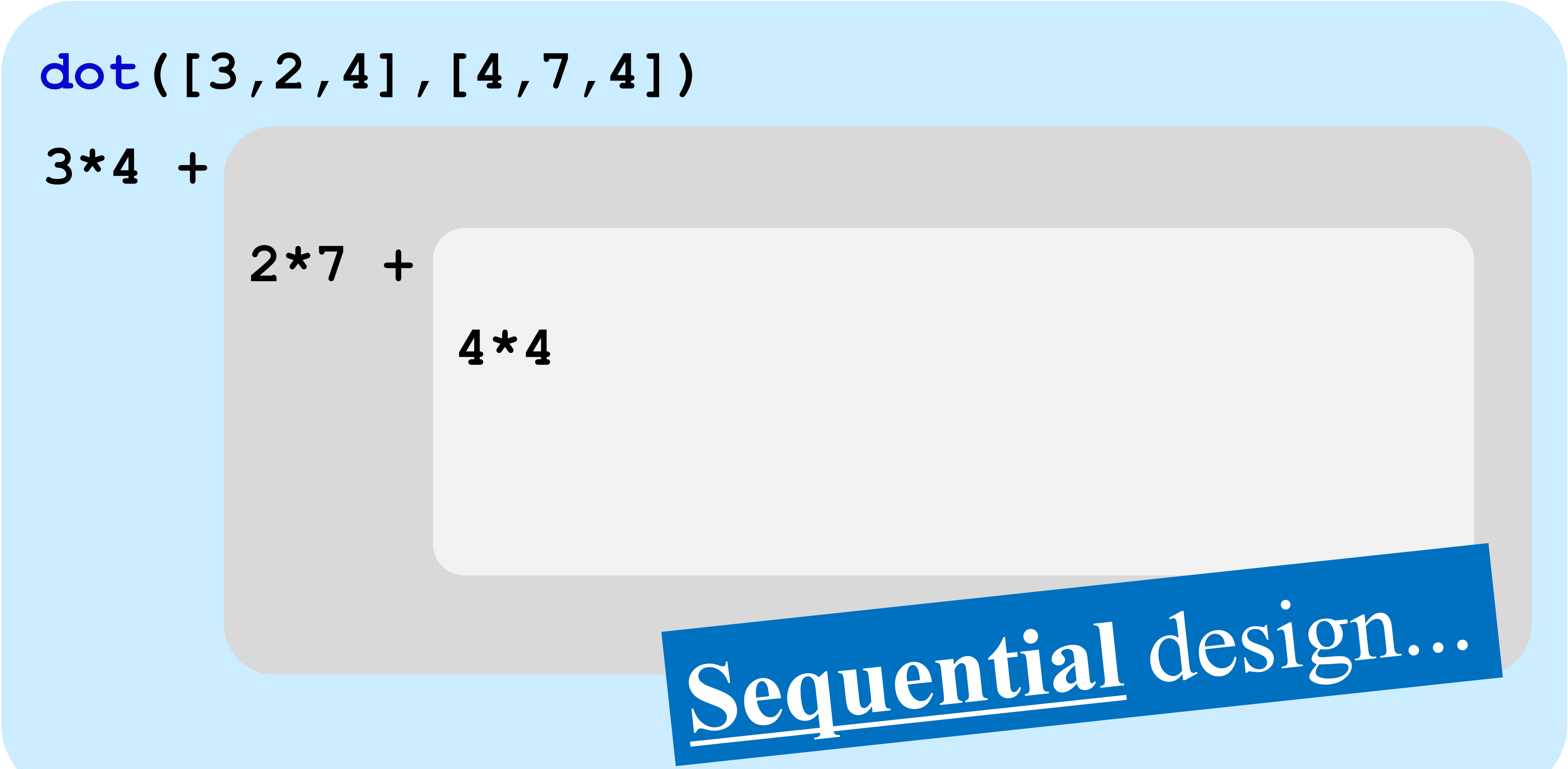
"Just last month I took a survey with no progress bar at the bottom of each page," she said. "I felt lost and confused. I'm glad there's a real

See **SURVEY** page 2

Are surveys the broccoli of our digital age?

Week 1, big-picture...

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

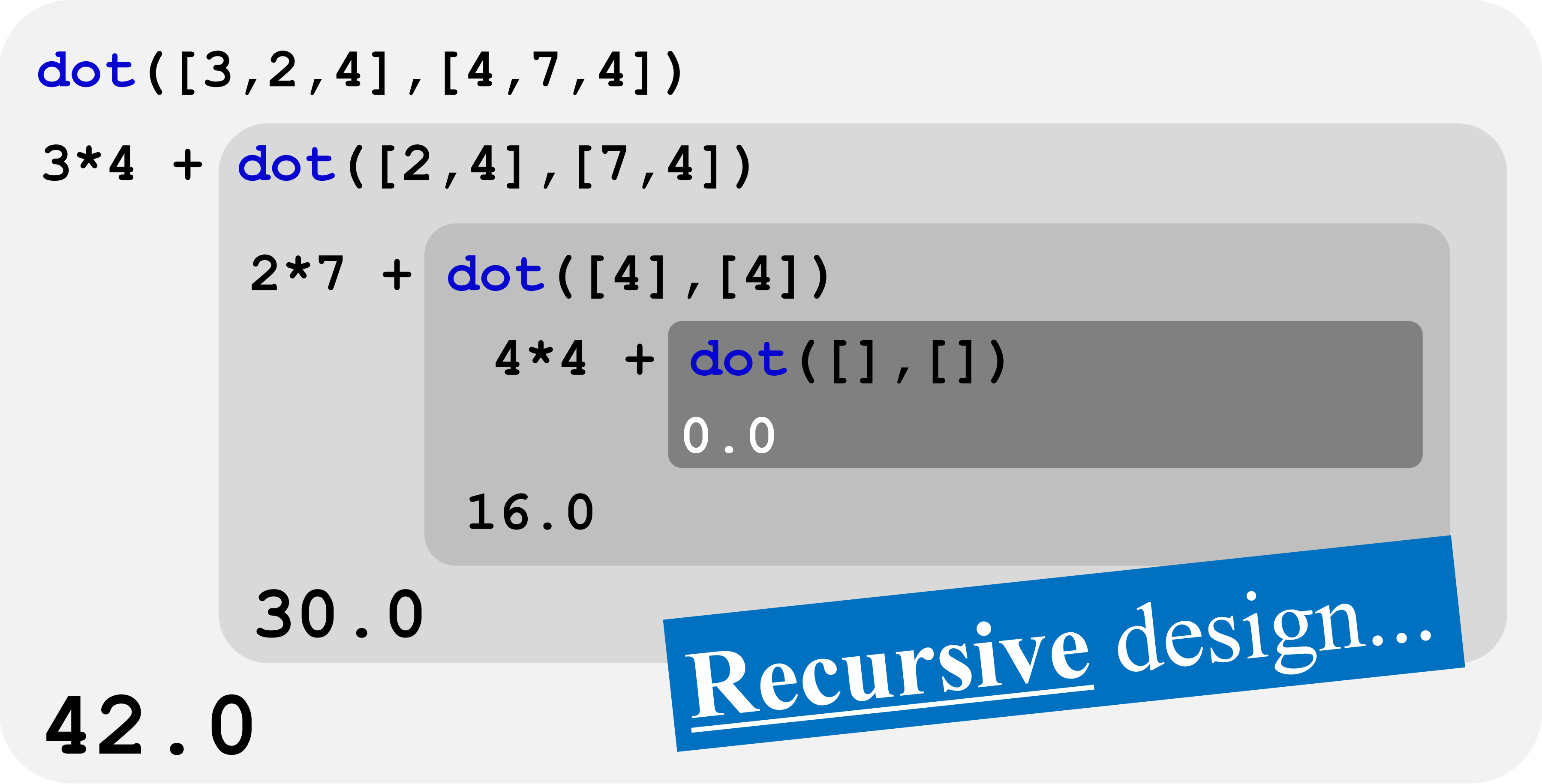


```
dot([3,2,4],[4,7,4])

3*4 +
        2*7 +
                4*4
```

Sequential design...

Dot product...recursively!

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



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

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

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

                4*4 + dot([],[])
                      0.0

              16.0

        30.0

42.0
```

Recursive design...

Recursion's idea:

We handle BASE + FIRST cases

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 BASE + FIRST cases

Recursion handles the REST

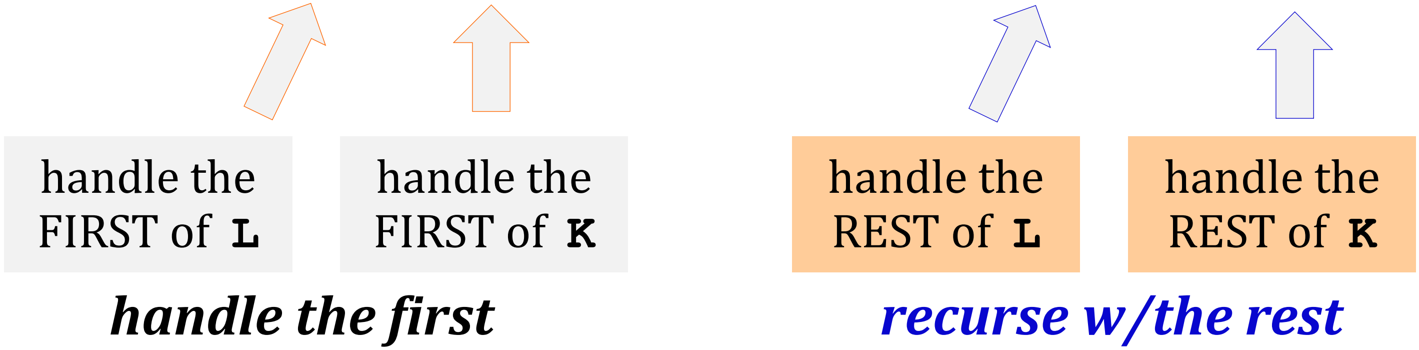
```
def dot( L, K ):
    if len(L) == 0 or len(K) == 0:
        return 0.0

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

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

Empty Cases

Base Cases

Specific/General case(s)

combine



handle the FIRST of **L**

handle the FIRST of **K**

handle the REST of **L**

handle the REST of **K**

handle the first

recurse w/the rest

Dot product... recursively! With code!

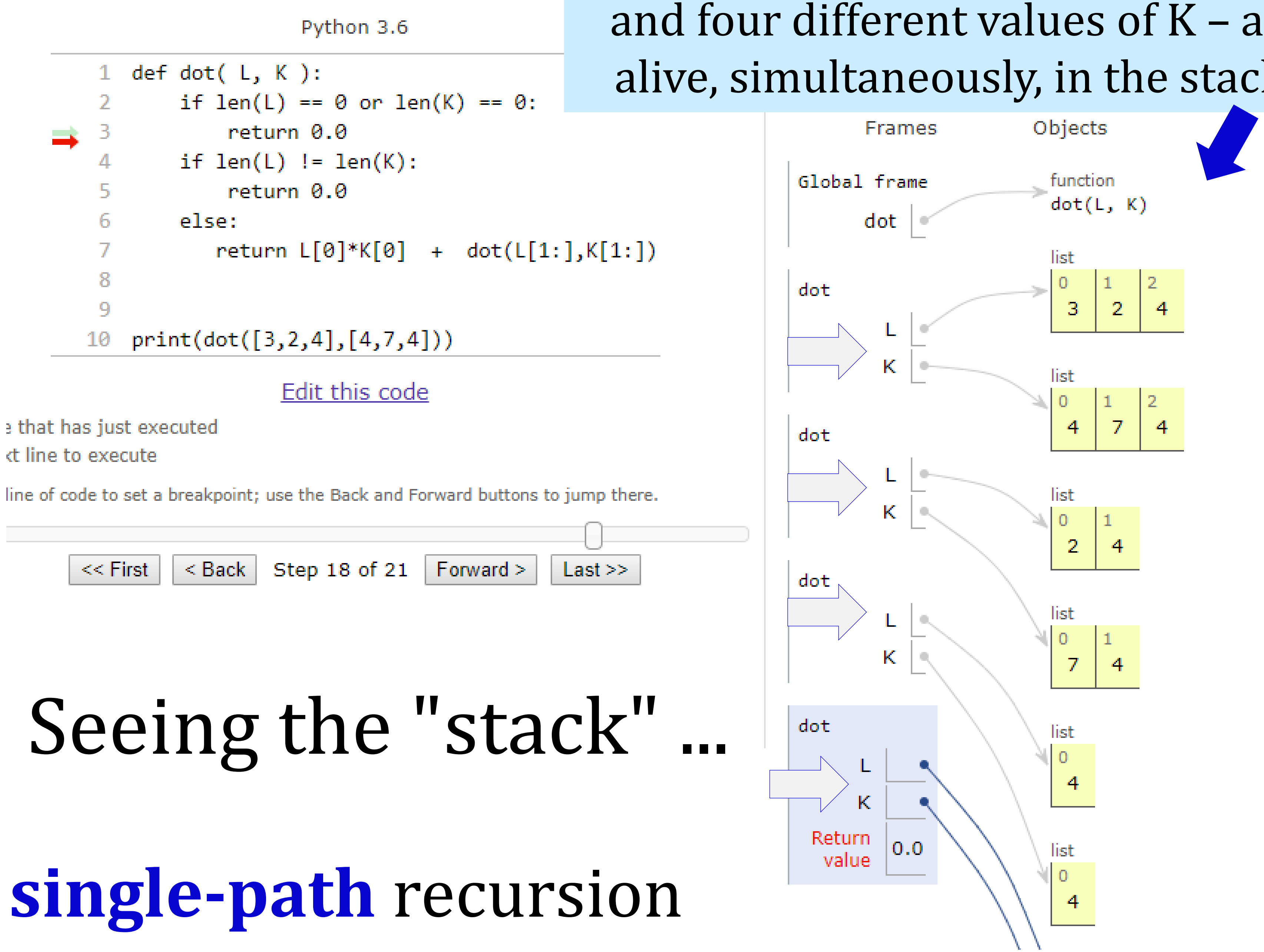
```
def dot( L, K ):
    if len(L) == 0 or len(K) == 0:
        return 0.0
    if len(L) != len(K):
        return 0.0
    else:
       return L[0]*K[0]  +  dot(L[1:],K[1:])
```

```
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
```

slow and steady!

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]))
```

Edit this code

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" ...

single-path recursion

A *random* aside: **Libraries!**

```
from random import *
```

```
choice( L )
```

don't need to use the library name

all random functions are now available!

---

requires using the library name

```
import random
```

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

A *random* aside: **Libraries!**

```
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: **Libraries!**

```
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: **Libraries!**

```
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]
```

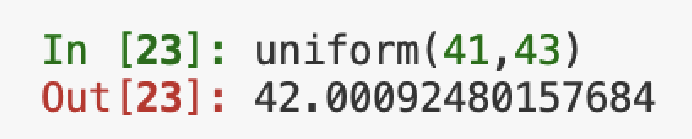
range

```
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:

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

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:

[0,1,2,3,4]

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

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

What's most likely?

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

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

More likely even or odd?

Watch out!

```
choice( '1,2,3,4' )
c
cl
un
```

being > 0?

Let's try it!

```
cho (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...

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:

[0,1,2,3,4]

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

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

What's most likely?

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

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

More likely even or odd? (0 is even)

odd or even?

Watch out!

```
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?

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

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

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

Syntax corner...

Which 2 of these 3 are *syntax errors*?

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

Solutions!

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**

[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]

Watch out!

`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**

Syntax corner…

`choice(0,1,2,3,4)` — **syntax error:** needs list [...] or str '...'

Which **2** of these 3 are *syntax errors*?

`choice([list(range(5))])` — **correct:** ***always*** **returns [0,1,2,3,4]** — **1/1 chance**

syntactically correct – actually *do*?

`choice[list(range(5))]` — **syntax error:** needs parens: choice(...)

The two *Monte Carlos*

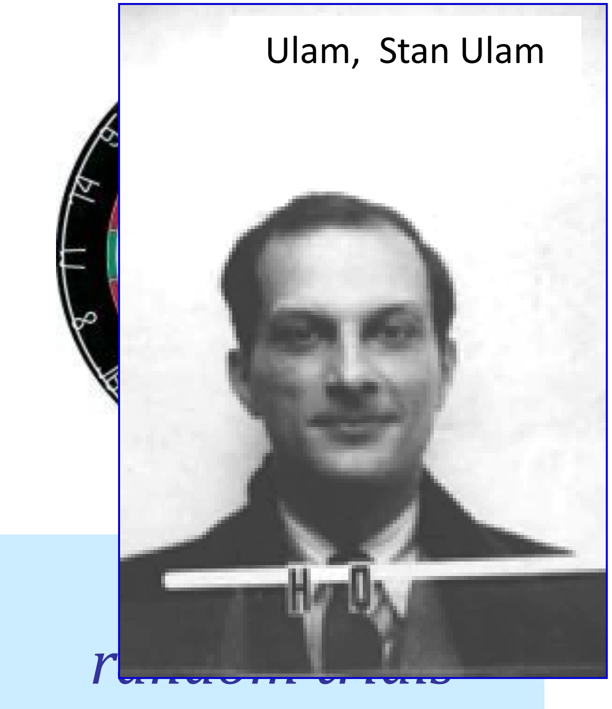
Monte Carlo Casino, **Monaco**

Insights via
random trials

Monte Carlo
Methods, **Math/CS**

The two *Monte Carlos*

Monte Carlo casino, **Monaco**



Monte Carlo
methods, **Math/CS**

A *"Monte Carlo"* function...

```
from random import *

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

    if compguess == hidden:
        return 1
    else:
        return 1 + guess( hidden )
```

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

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

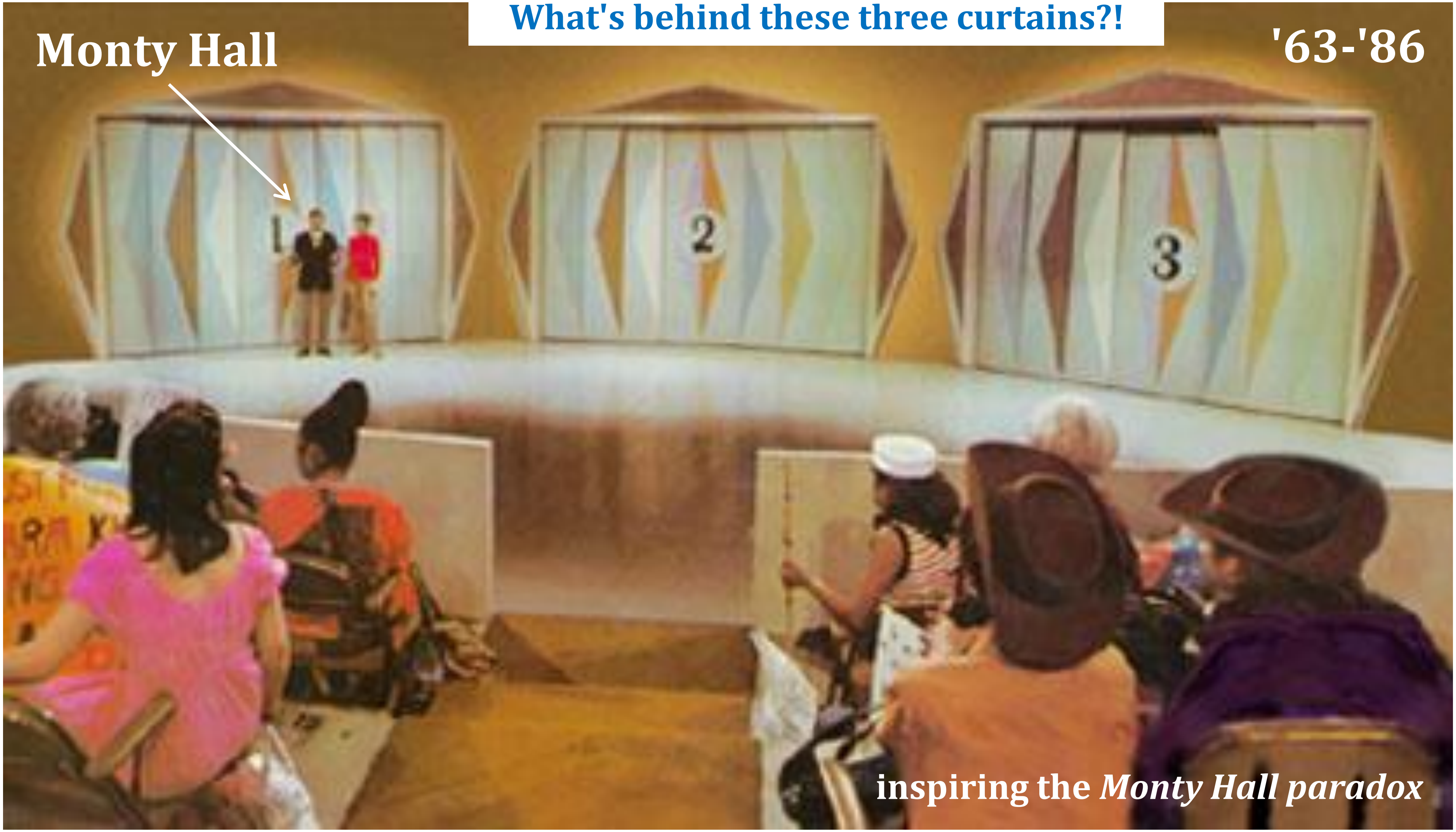
```
def countDoubles( N ):
    """  input: the # of dice rolls to make
        output: the # of doubles seen """
    if N == 0:
        return 0       # zero rolls, zero doubles…
    else:
        d1 = choice( [1,2,3,4,5,6] )
        d2 = choice( list(range(1,7)) )

        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?

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

doubles check?

Monte Carlo *Curtains*



What's behind these three curtains?!

Monty Hall

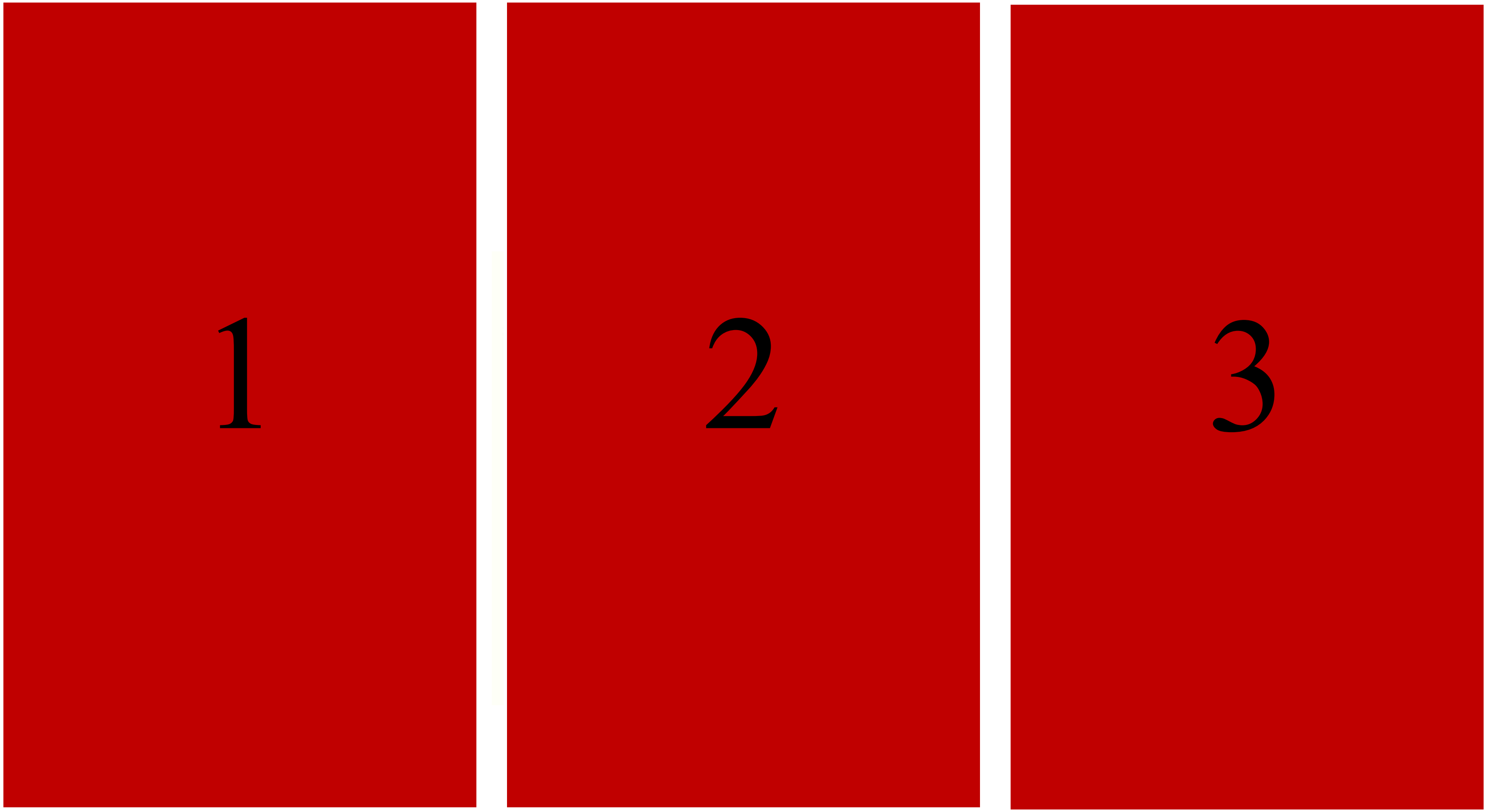
'63-'86

inspiring the *Monty Hall paradox*

Monte Carlo Monty Hall

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

Monte Carlo Monty Hall



1

2

3

Let's play!

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

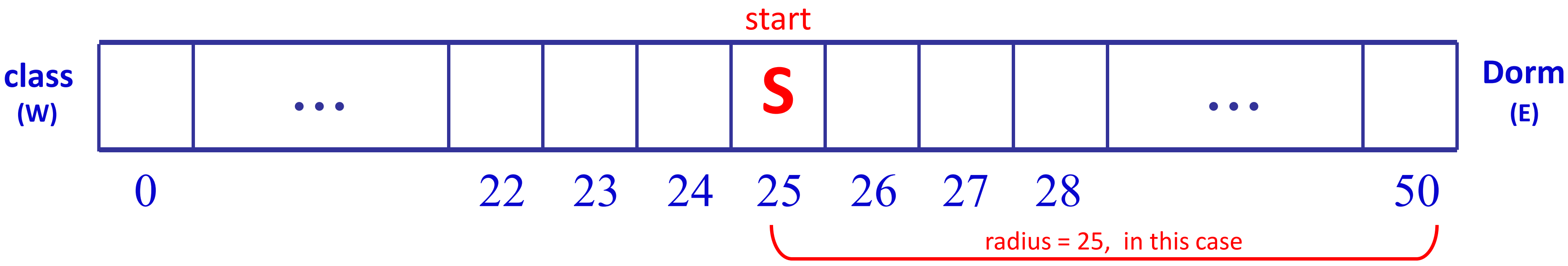
```
def MCMH( init, sors, N ):
  """ plays the "Let's make a deal" game N times
      returns the number of times you win the *Spam!*
  """
  if N == 0: return 0              # don't play, can't win
  przDoor = choice([1,2,3])        # where the spam (prize) is…

  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, ':', 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	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥

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

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)`

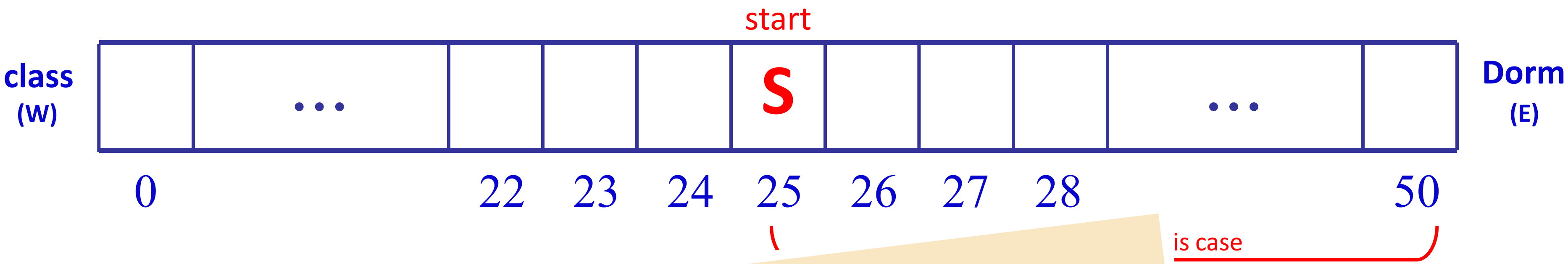
take **nsteps** random steps starting at **st**

`rwsteps(st,low,hi)`

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

An example *closer to home*

hw2pr2



is case

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...

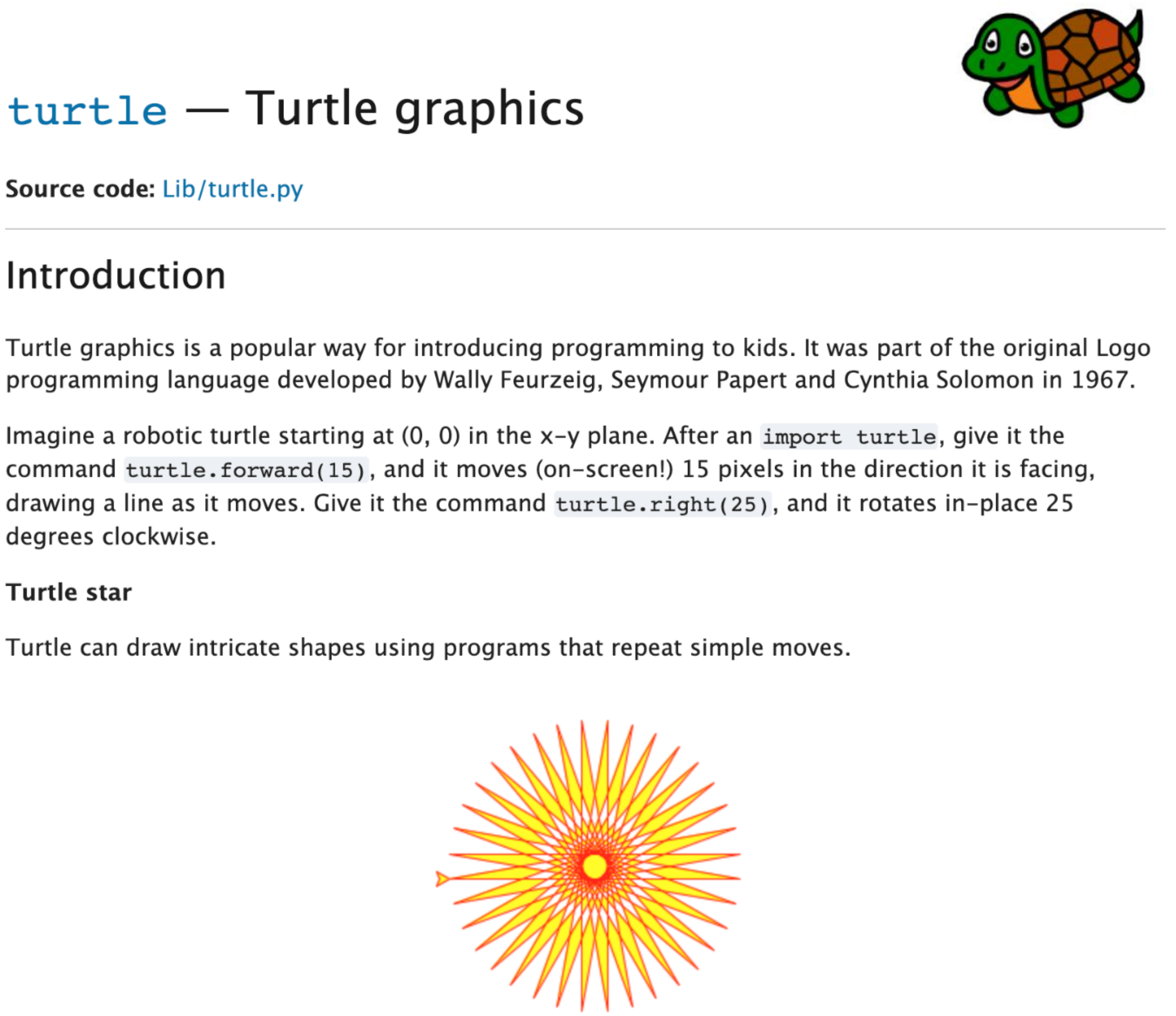
`eps)`

`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

Introduction

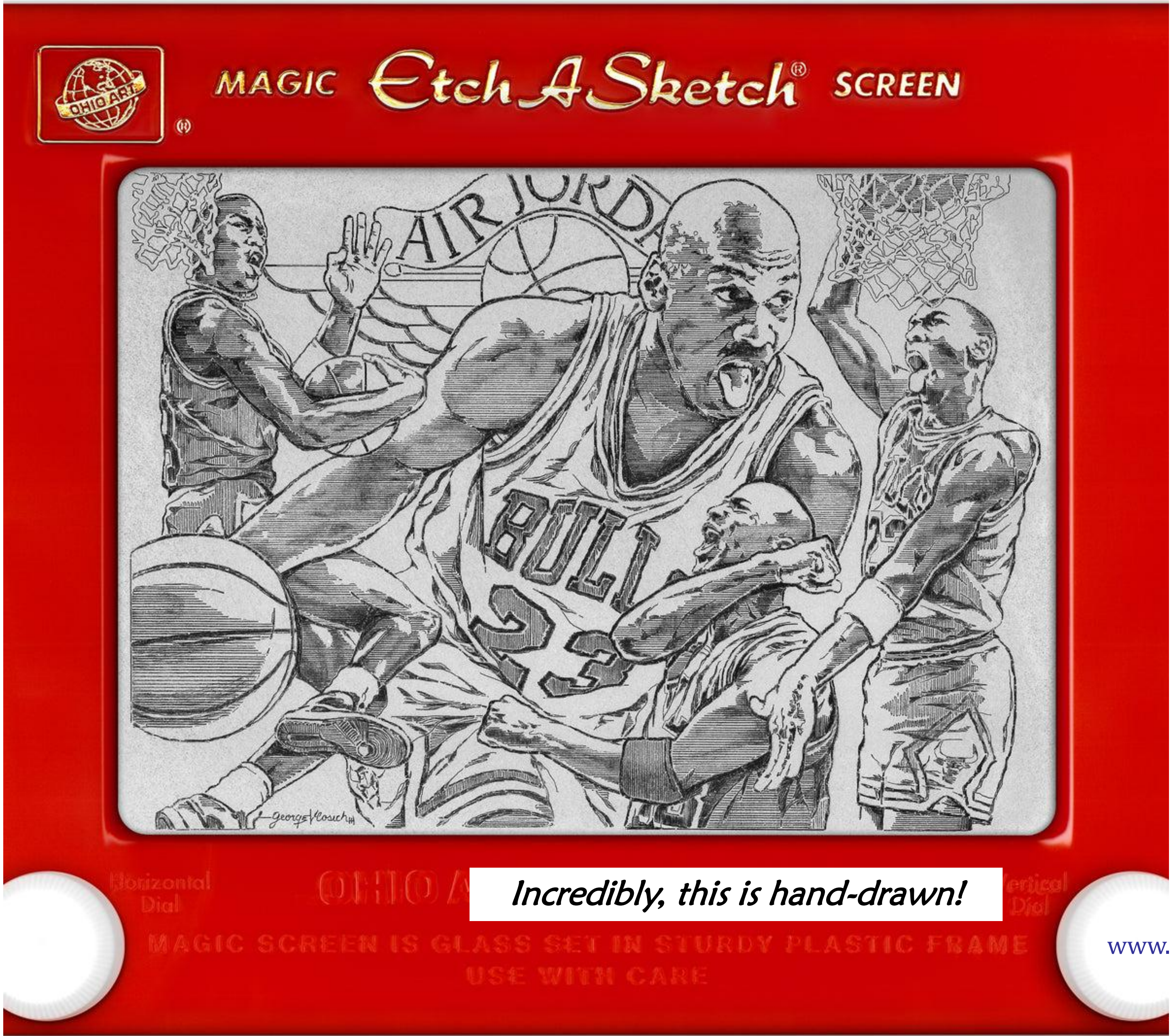
Turtle graphics is a popular way for introducing programming to kids. It was part of the original Logo programming language developed by Wally Feurzeig, Seymour Papert and Cynthia Solomon in 1967.

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.

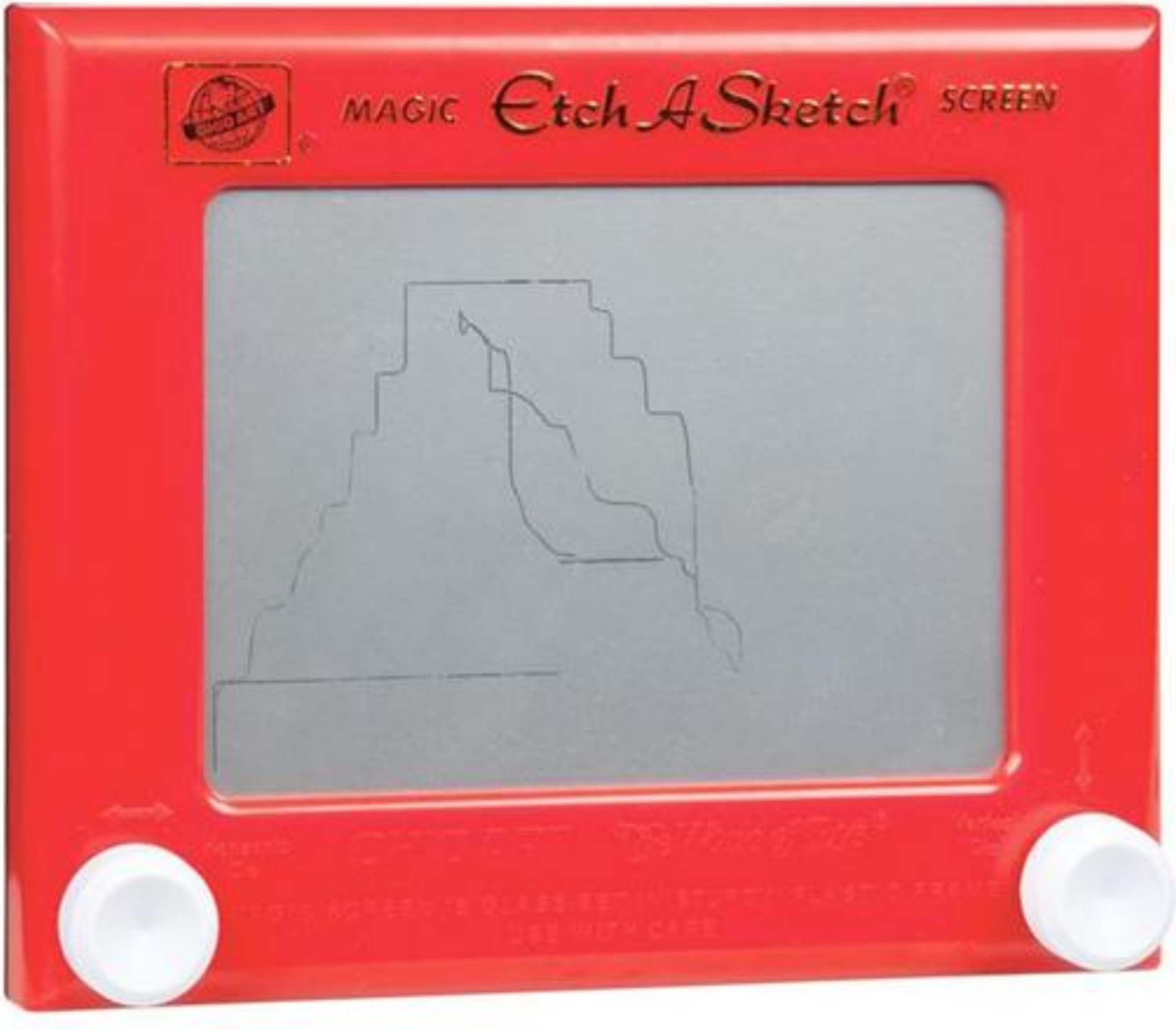
Lab 2 ~ *Python's Etch-a-Sketch*



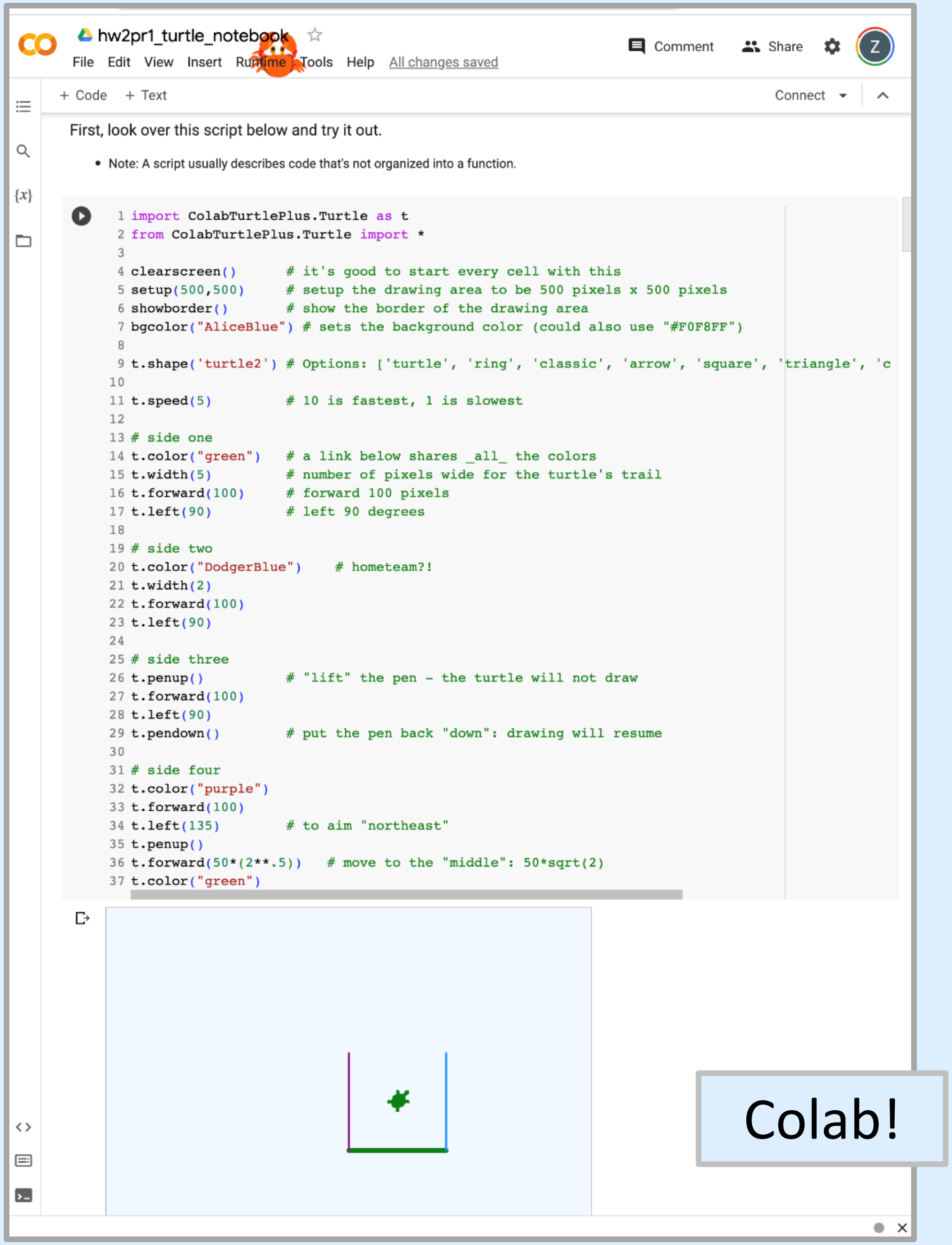
Incredibly, this is hand-drawn!

www.gvartwork.com

more *typical* etch-a-sketch result



In-browser Python...



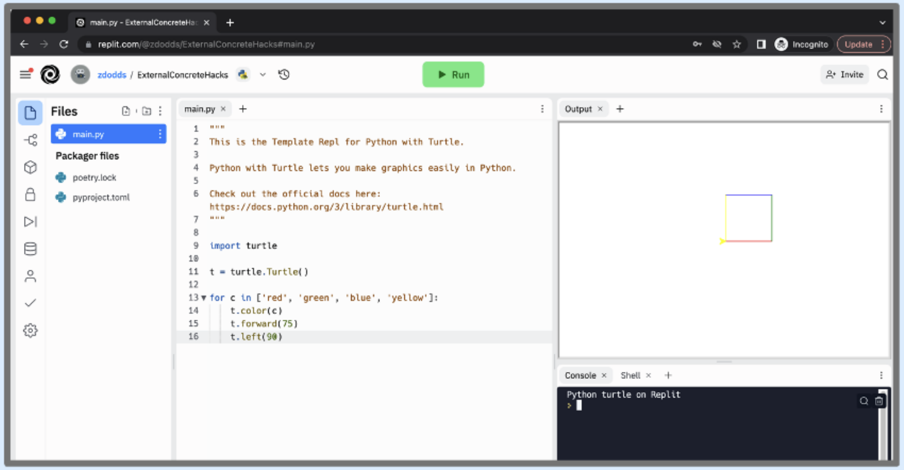
First, look over this script below and try it out.

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

```python
import ColabTurtlePlus.Turtle as t
from ColabTurtlePlus.Turtle import *

clearscreen()      # it's good to start every cell with this
setup(500,500)     # setup the drawing area to be 500 pixels x 500 pixels
showborder()       # show the border of the drawing area
bgcolor("AliceBlue") # sets the background color (could also use "#F0F8FF")

t.shape('turtle2') # Options: ['turtle', 'ring', 'classic', 'arrow', 'square', 'triangle', 'c

t.speed(5)         # 10 is fastest, 1 is slowest

# side one
t.color("green")   # a link below shares _all_ the colors
t.width(5)         # number of pixels wide for the turtle's trail
t.forward(100)     # forward 100 pixels
t.left(90)         # left 90 degrees

# side two
t.color("DodgerBlue")    # hometeam?!
t.width(2)
t.forward(100)
t.left(90)

# side three
t.penup()          # "lift" the pen - the turtle will not draw
t.forward(100)
t.left(90)
t.pendown()        # put the pen back "down": drawing will resume

# side four
t.color("purple")
t.forward(100)
t.left(135)        # to aim "northeast"
t.penup()
t.forward(50*(2**.5))    # move to the "middle": 50*sqrt(2)
t.color("green")
```

Colab!

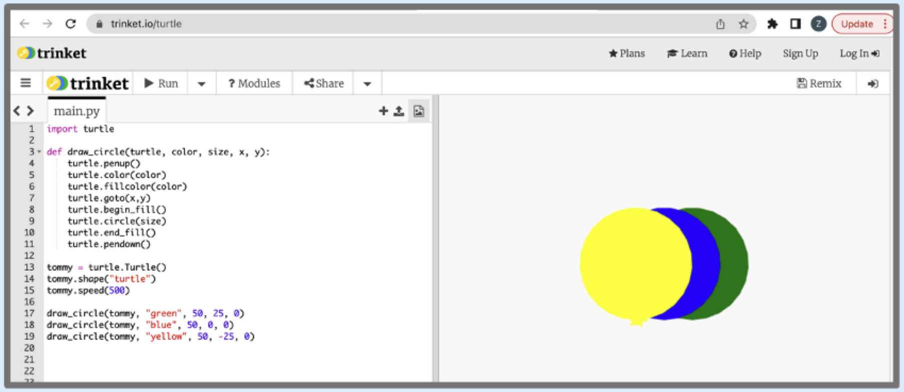
In-browser alternatives…


replit.com/@zdodds/ExternalConcreteHacks#main.py
zdodds / ExternalConcreteHacks
Run
Invite
Files
main.py
Packager files
poetry.lock
pyproject.toml
Output
Console
Shell

```python
"""
This is the Template Repl for Python with Turtle.

Python with Turtle lets you make graphics easily in Python.

Check out the official docs here:
https://docs.python.org/3/library/turtle.html
"""

import turtle

t = turtle.Turtle()

for c in ['red', 'green', 'blue', 'yellow']:
    t.color(c)
    t.forward(75)
    t.left(90)
```

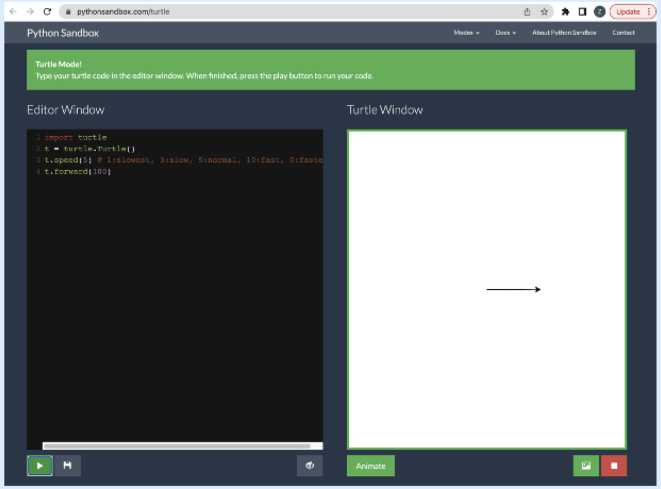
```
Python turtle on Replit
>
```

repl.it



```python
import turtle

def draw_circle(turtle, color, size, x, y):
    turtle.penup()
    turtle.color(color)
    turtle.fillcolor(color)
    turtle.goto(x,y)
    turtle.begin_fill()
    turtle.circle(size)
    turtle.end_fill()
    turtle.pendown()

tommy = turtle.Turtle()
tommy.shape("turtle")
tommy.speed(500)

draw_circle(tommy, "green", 50, 25, 0)
draw_circle(tommy, "blue", 50, 0, 0)
draw_circle(tommy, "yellow", 50, -25, 0)
```

Trinket



Turtle Mode!
Type your turtle code in the editor window. When finished, press the play button to run your code.

```python
import turtle
t = turtle.Turtle()
t.speed(5) # 1:slowest, 3:slow, 5:normal, 10:fast, 0:faste
t.forward(100)
```

Python sandbox

Single-path recursion

A starter *script*:

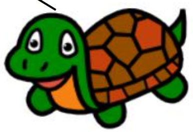
```
# a triangle
# as a _script_
forward(100)
left(120)
forward(100)
left(120)
forward(100)
left(120)
```

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

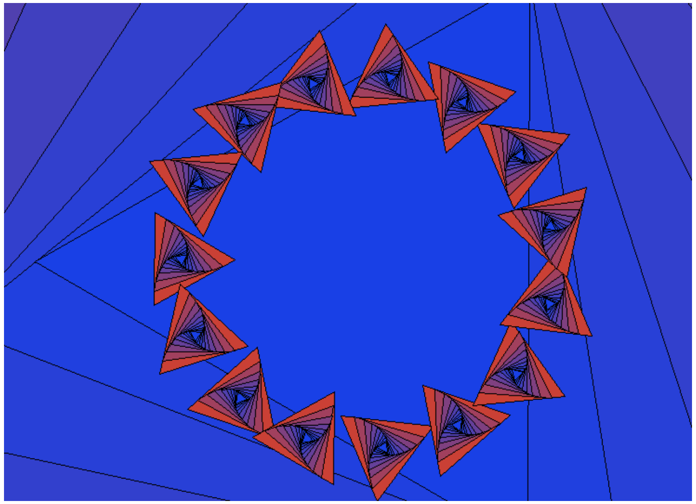
And a starter *function*:

```
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)
```

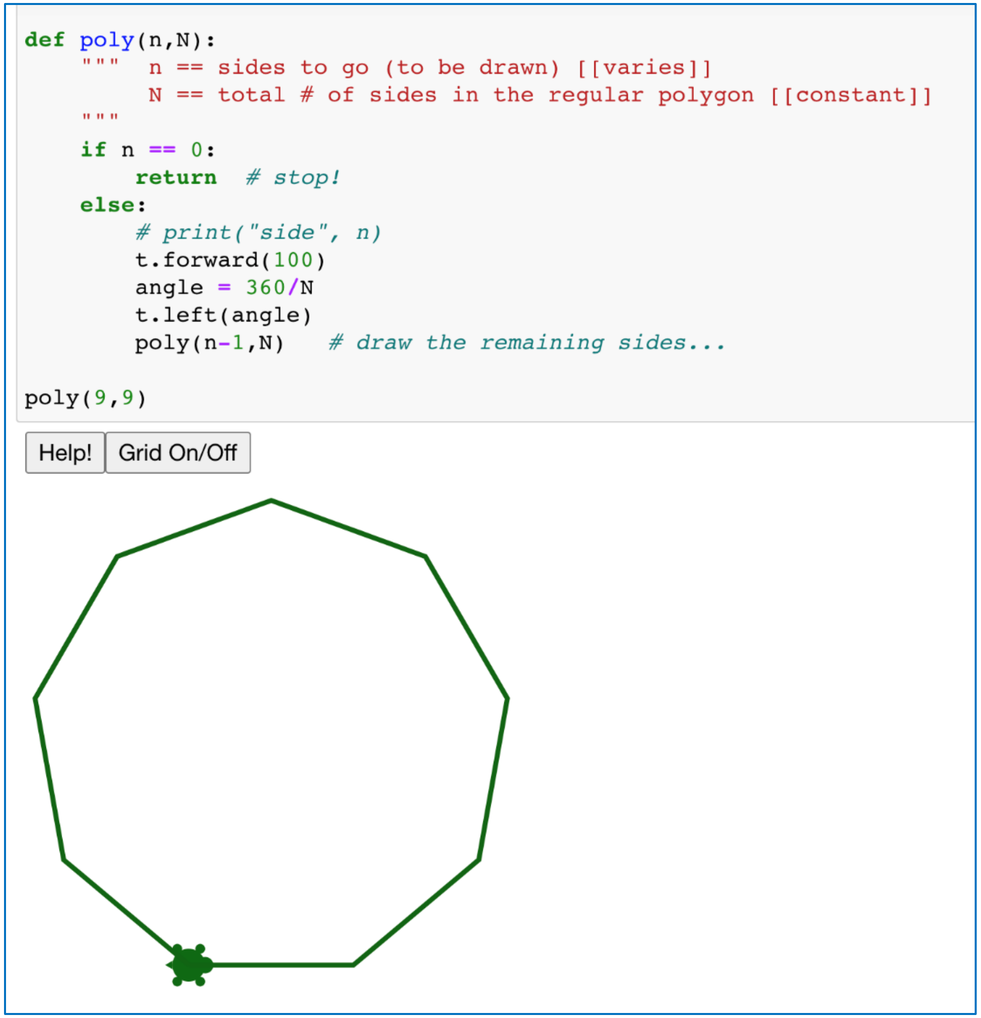
I don't know about **tri**, but there's *no* **return** ... !



Turtle's ability?
It varies...



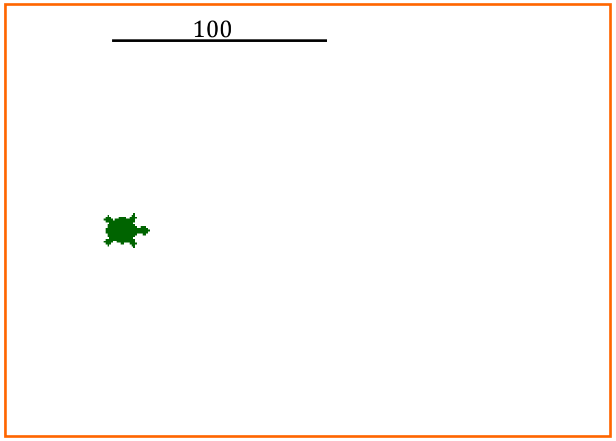
it can vary *widely*



```
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)
```

Be the turtle !

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

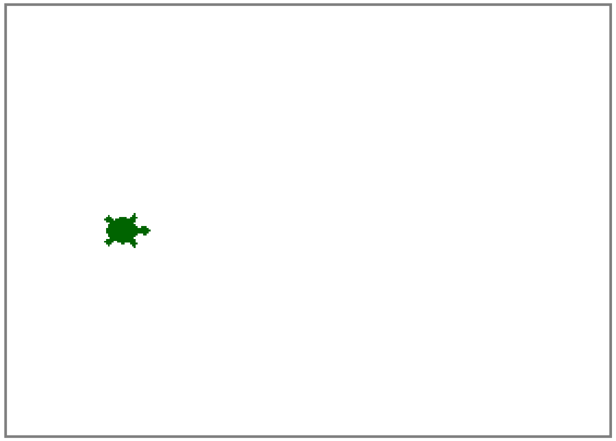


```
def chai(dist):
    """ mystery fn! """
    if dist < 20:
        return
    else:
        forward(dist)
        left(90)
        forward(dist/2.0)
        right(90)
        # line (a)
        right(90)
        forward(dist)
        left(90)
        # line (b)
        left(90)
        forward(dist/2.0)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

(2a)

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

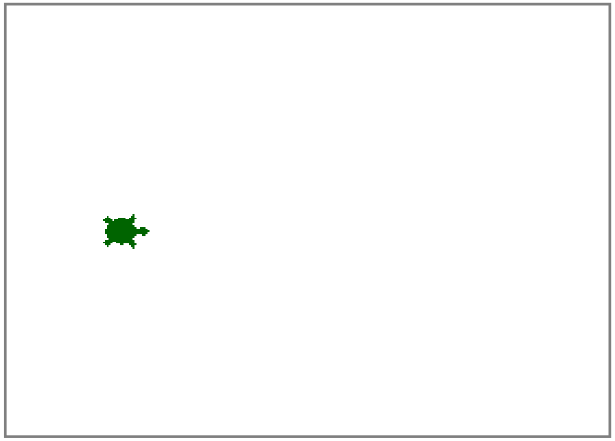
What would **chai(100)** then draw?



(2b)

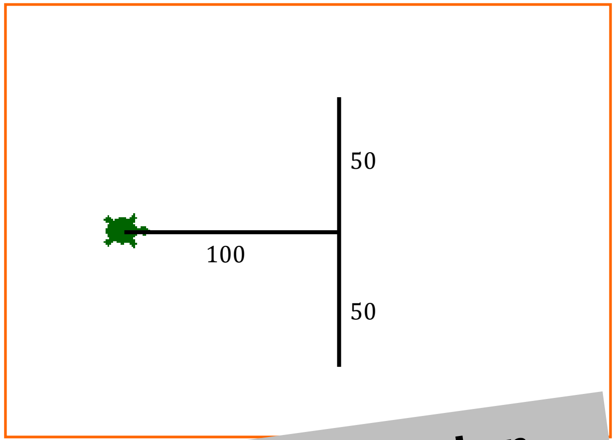
then, ALSO replace
`# line (b)`
with the same line:
`chai(dist/2)`

What would **chai(100)** *then* draw?!



Be the turtle !

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



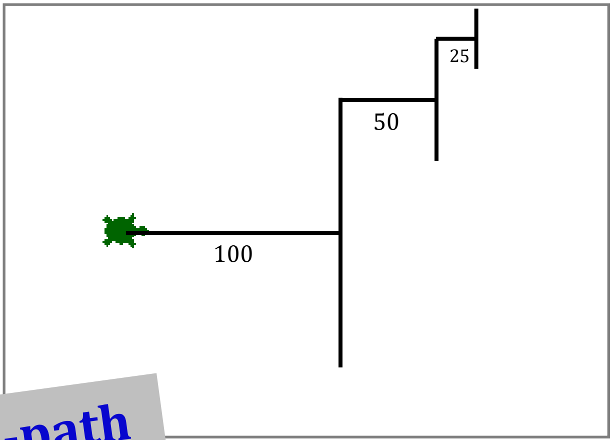
no recursion

```
def ch
    ""   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)
```

(2a)

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

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

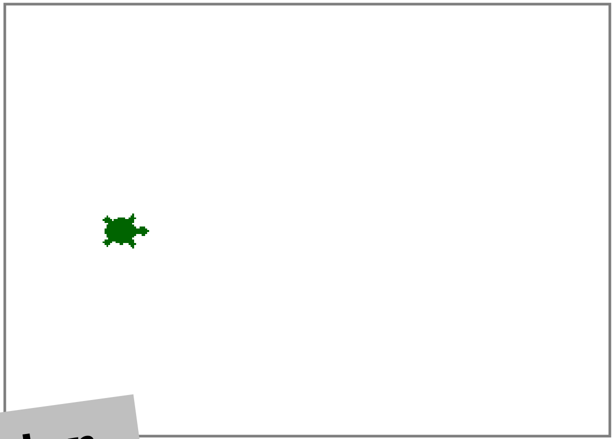


single-path recursion

(2b)

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

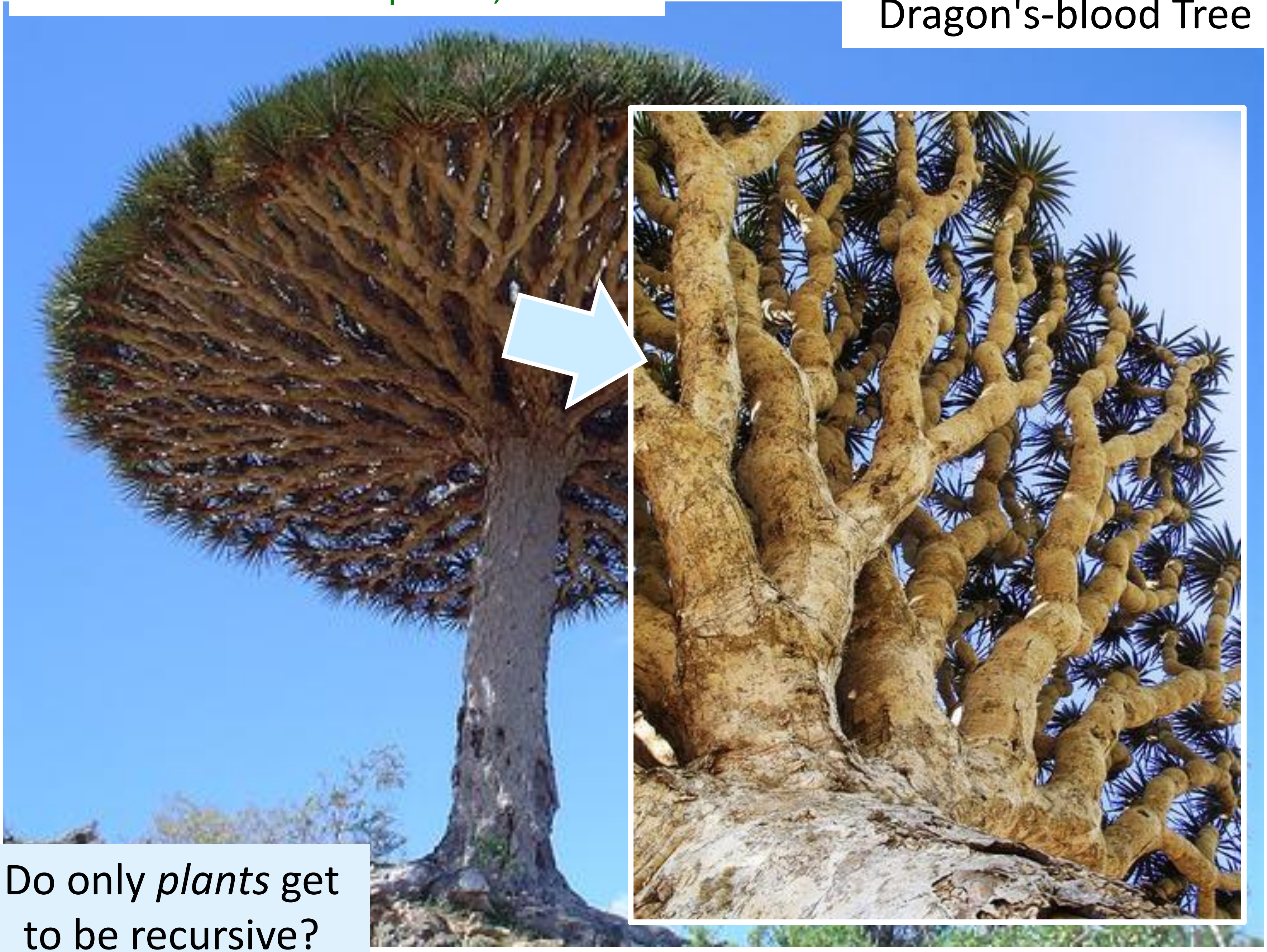
What would **chai(100)** *then* draw?!



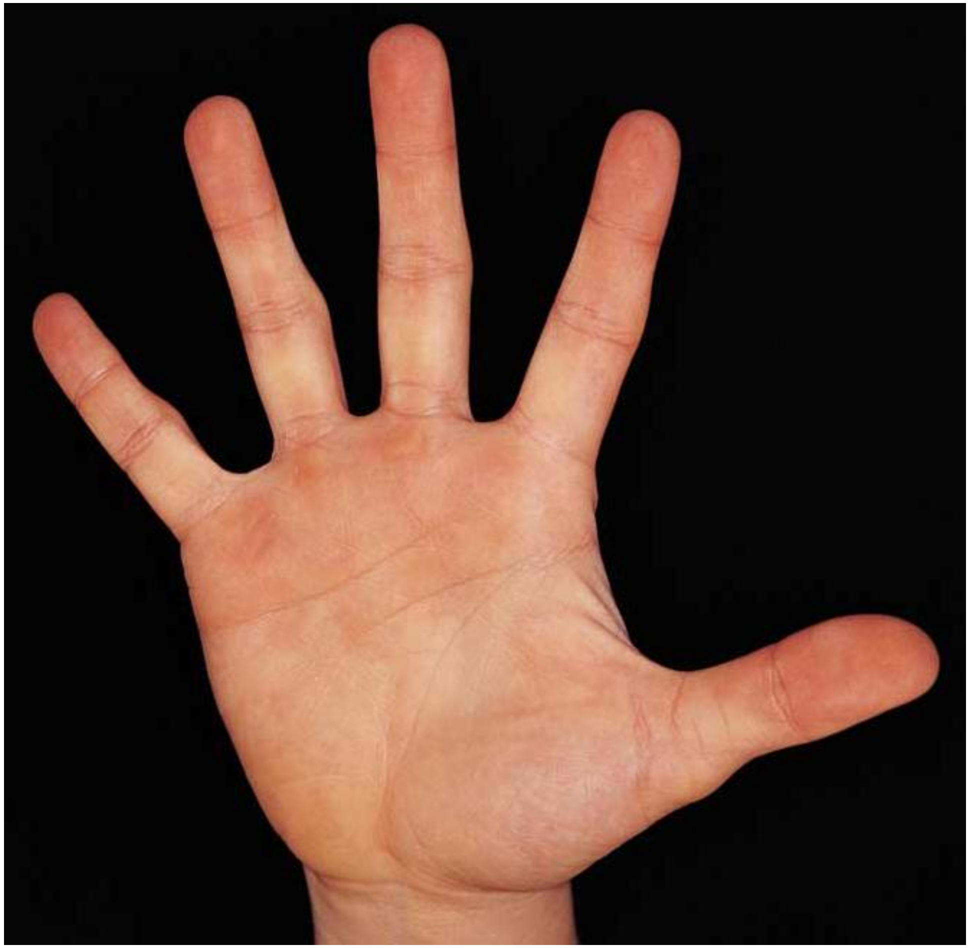
branching recursion

A brief word from our sponsor, Nature...

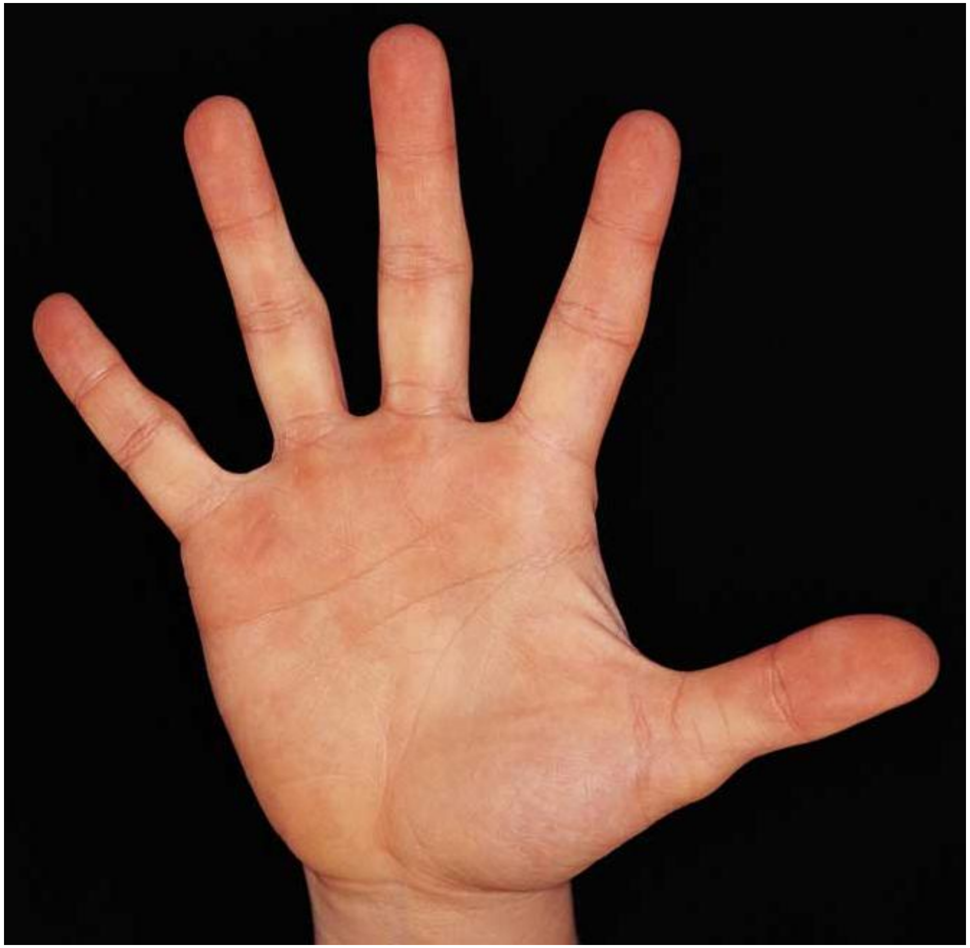
Dragon's-blood Tree



Do only *plants* get to be recursive?



Branching seems to be *plants-only*?



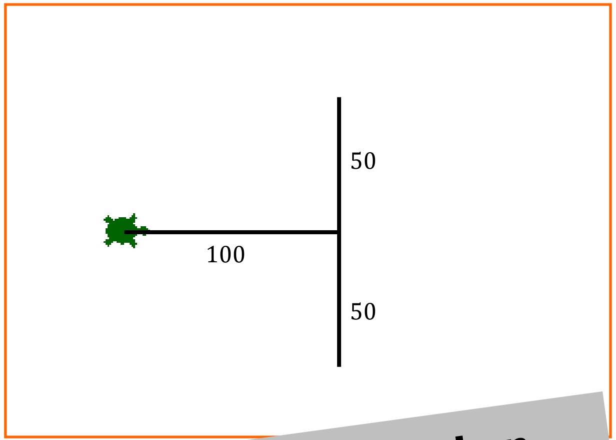
Branching recursion is *Strange*!

Br tr ger

one layer more!

Be the turtle !

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



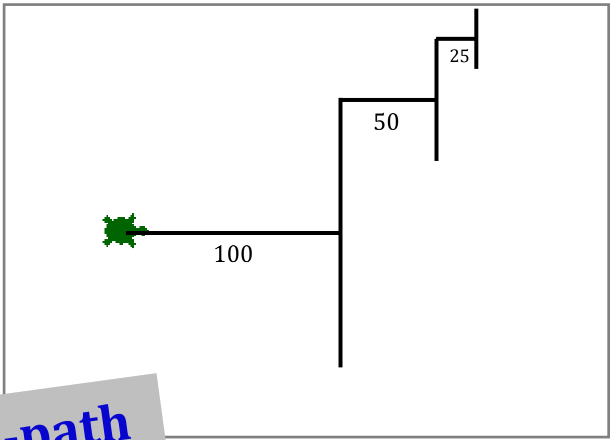
no recursion

```
def ch
    ""   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)
```

(2a)

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

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

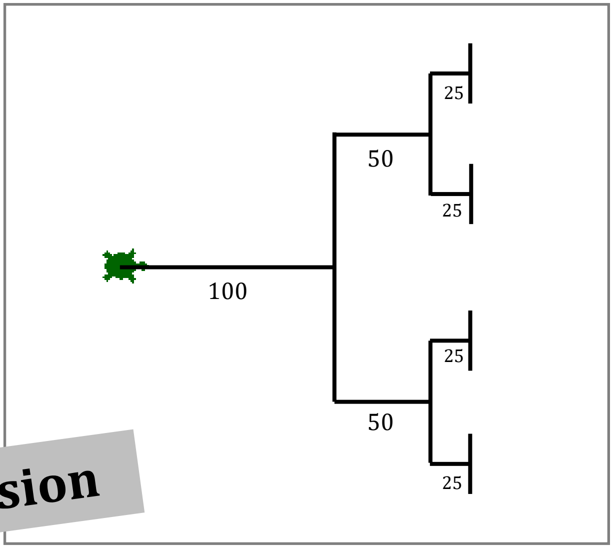


single-path recursion

(2b)

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

What would **chai(100)** *then* draw?!

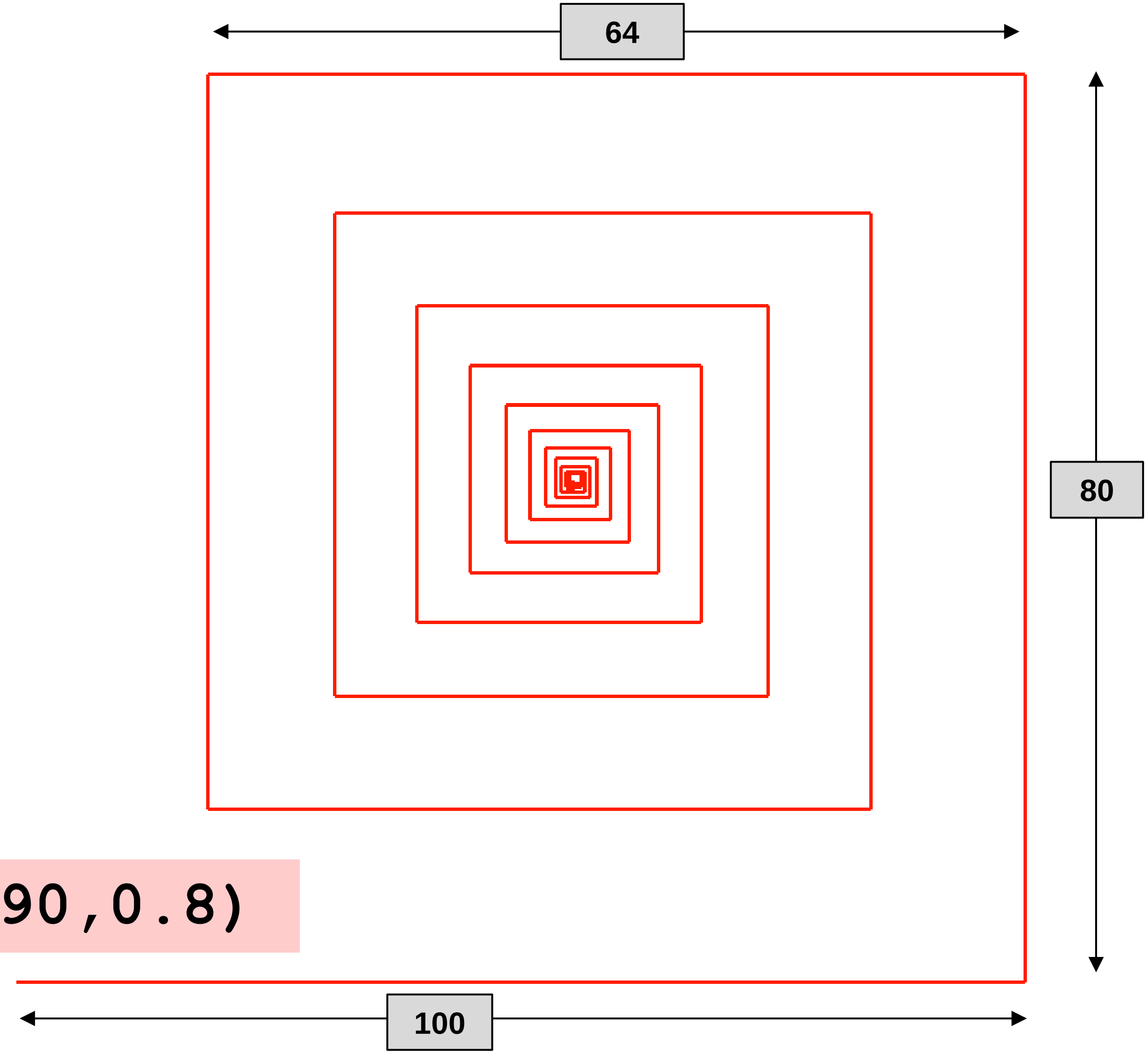


branching recursion

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

lab ~ hw2pr1

fractal art

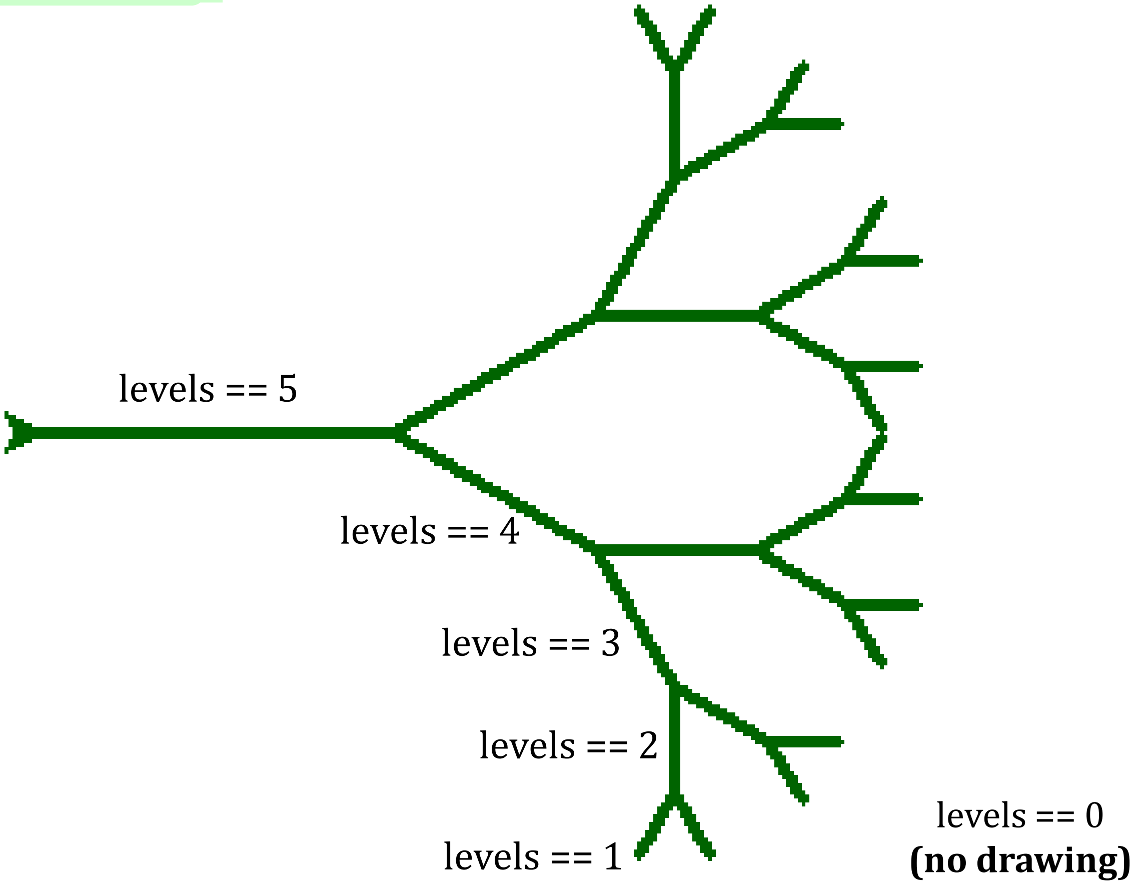


```
spiral(100,90,0.8)
```

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

`svtree( trunkLength, levels )`

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



levels == 5

levels == 4

levels == 3

levels == 2

levels == 1

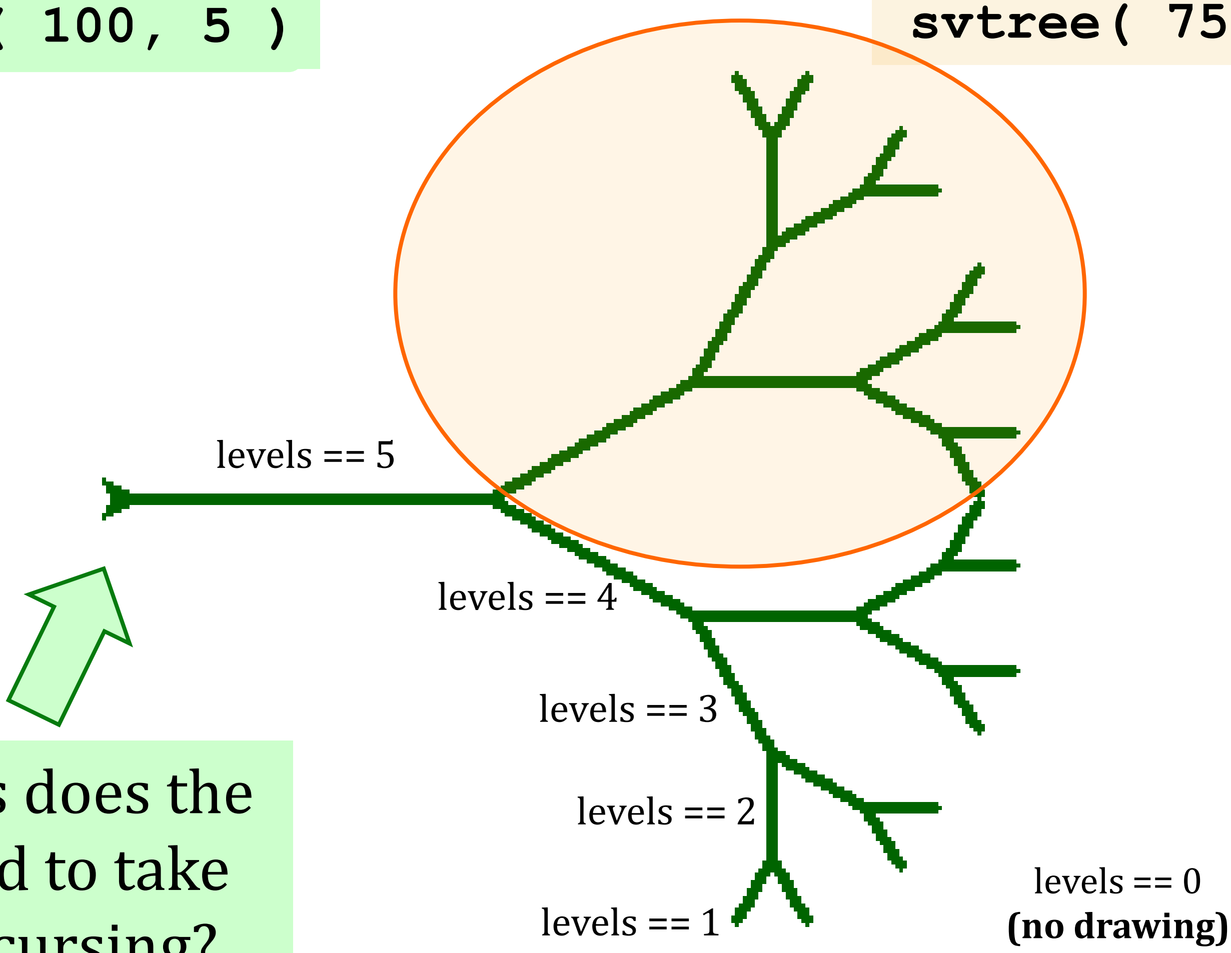
levels == 0
(no drawing)

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

`svtree( trunkLength, levels )`

`svtree( 100, 5 )`

`svtree( 75, 4 )`

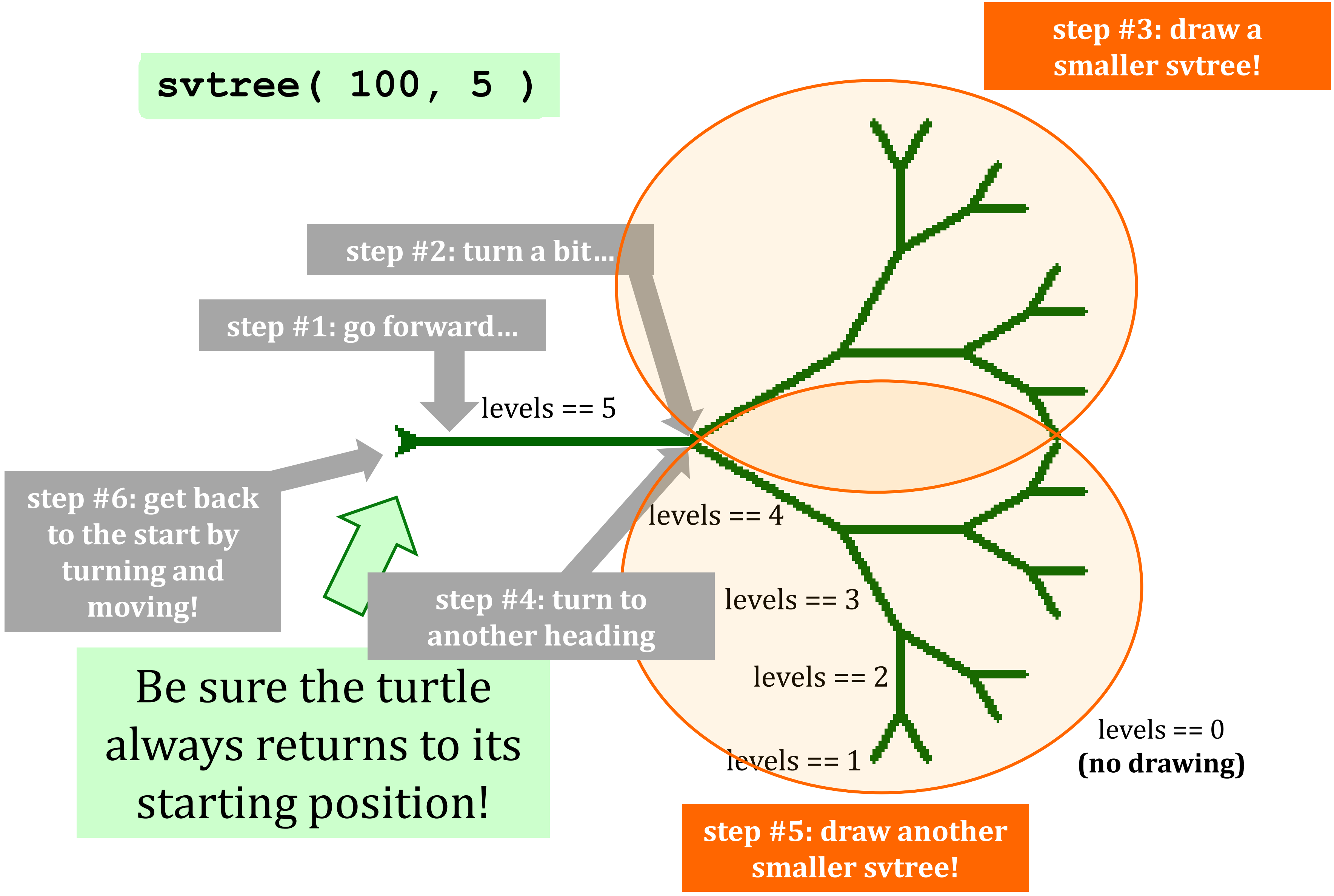


What steps does the turtle need to take before recursing?

*Branching **recursion!***

`svtree( trunkLength, levels )`

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



step #1: go forward…

step #2: turn a bit…

step #3: draw a smaller svtree!

step #4: turn to another heading

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!

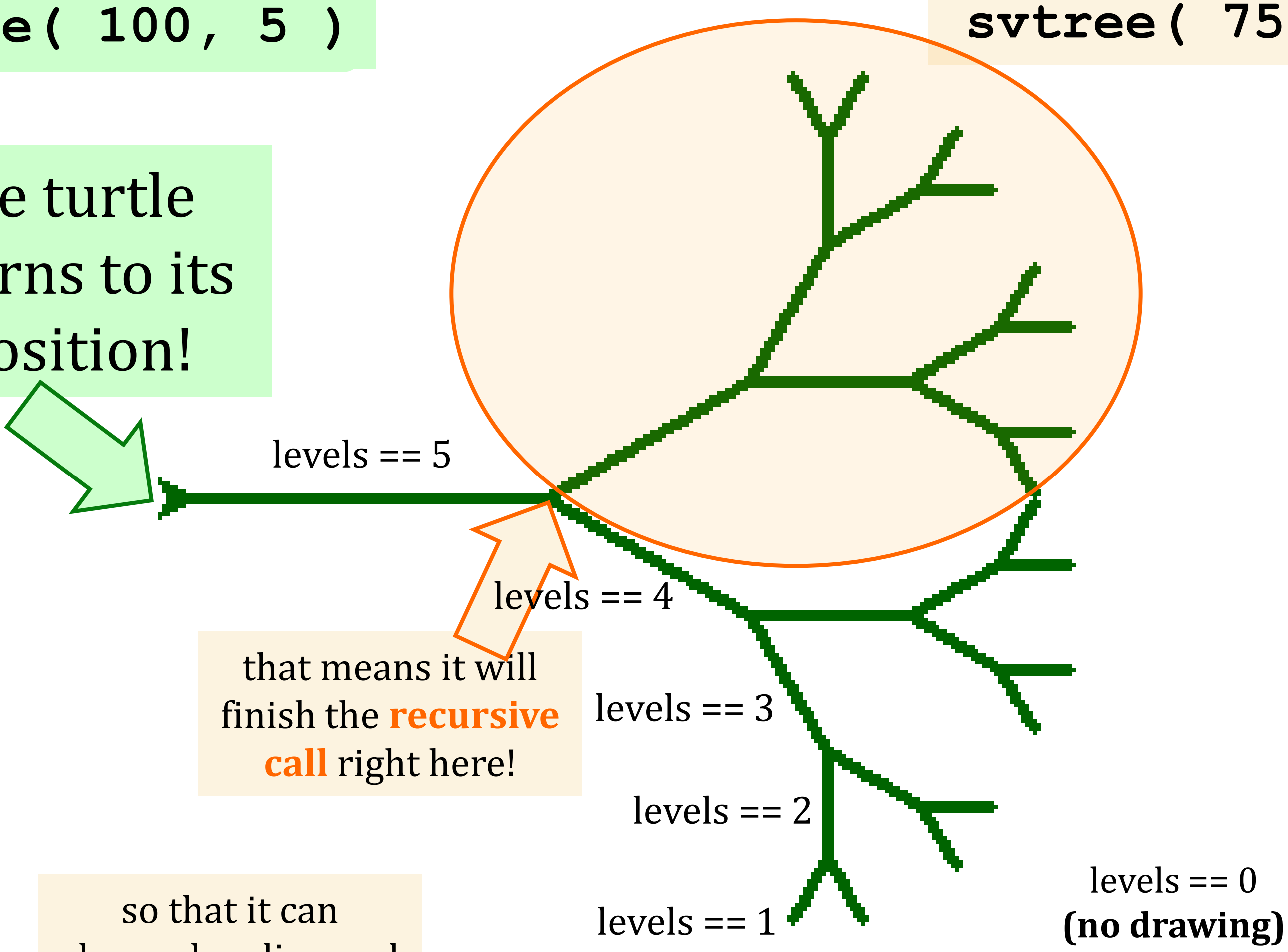
Branching recursion!

svtree(trunkLength, levels)

svtree(100, 5)

svtree(75, 4)

Be sure the turtle always returns to its starting position!

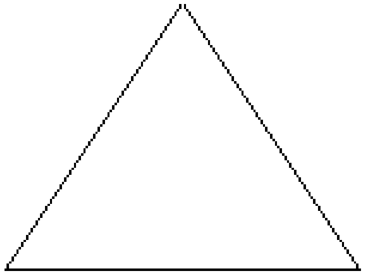


that means it will finish the **recursive call** right here!

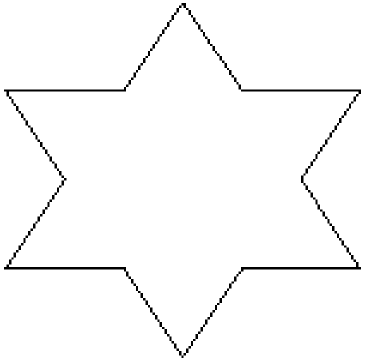
so that it can change heading and draw another one…

***Branching** recursion!*

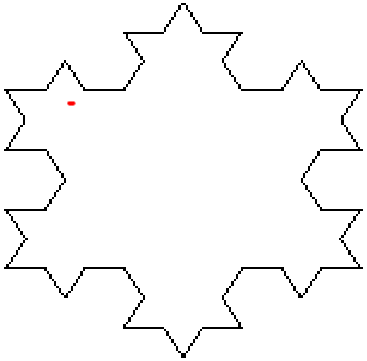
The Koch curve



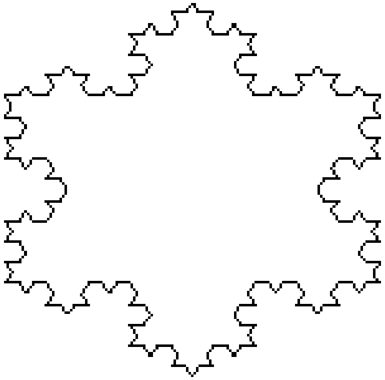
snowflake(100, 0)



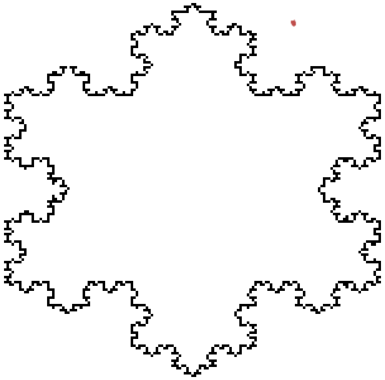
snowflake(100, 1)



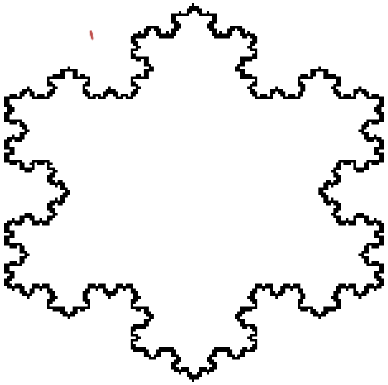
snowflake(100, 2)



snowflake(100, 3)



snowflake(100, 4)



snowflake(100, 5)

Single-path or Branching recursion here?