Cyriak: conceptually disruptive recursion...

*Fractals and Turtles*

Baaa

CS 5 alien on strike!

CS 5 green mascot representing today's terrestrial theme

CS 5 Today

**hw3** due Monday...

Lots of tutoring...

How random!
Recursive painting...

https://nubleh.github.io/i_painted/
Recursion's idea:

```python
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**
- You handle the FIRST
- Recursion handles the REST

**Combining Elements**
- Handle the FIRST of L
- Handle the FIRST of K
- Handle the REST of L
- Handle the REST of K

**Firsts, as appropriate**
**Recursion with the rest**

**First**
**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:])

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

Seeing the "stack" ...
Using a Euro, Yuan the deadline comes next time I’ll try to make the Mark. I’m Pounding it into my brain, but I have to Peso much attention to other deadlines in the classes I Currency have - but if this keeps up, my grade will be demolished to Ruble.
import random
from random import *

allows use of `dir(random)` and `help(random)`
all random functions are now available!
Some *random* asides...

```python
import random
from random import *

choice(L)  # chooses 1 element from the sequence L
choice('mudd')  # ... or 1 character from a string
choice(['cmc', 'scripps', 'pitzer', 'pomona'])
```

allows use of `dir(random)` and `help(random)`
all random functions are now available!
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'])

list(range(5))  \rightarrow \ [0,1,2,3,4]
list(range(1,5)) \rightarrow \ [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...

```python
from random import *

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

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

Remember, this is [0,1,...,99]
```

print the guesses?
slow down...
return the number of guesses?
investigate *expected* # of guesses?!!?
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 )
A few random thoughts...

```
from random import *

choice( [1,2,3,2] )

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

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

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

choice( [1,2,3,2] )

choice(list(range(5)))

uniform( -20.5, 0.5 )

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

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

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

What are the chances this returns a 2?

What are the chances of this returning a 4?

Careful!

What's the most likely return value here?

What's the most likely return value here?

What's the most likely return value here?

Is this more likely to be even or odd?

What're the chances of this being > 0? Extra!

Which two 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.

```python
from random import *

choice([1,2,3,2])  # What are the chances this returns a 2? 2/4 or 50%

choice(list(range(5)) + [4,2,4,2])  # What are the chances of this returning a 4? 3/9

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

choice(list(range(5)))  # Is this more likely to be even or odd? even, 3/5

uniform(-20.5, 0.5)  # What're the chances of this being > 0? 1/42

choice(0,1,2,3,4)  # syntax error: needs [...] or '...' 1/1 chance

choice([list(range(5))])  # correct: always returns [0,1,2,3,4]

choice[list(range(5))]  # syntax error: needs choice( ...) 1/1 chance
```

Team up and try this on the backpage first…
from random import *

choice([1,2,3,2]) —— What are the chances this returns a 2? 2/4 or 50%

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

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

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

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

choice(list(range(5)))]

syntax error: needs [... ] or '...' 
correct: always returns [0,1,2,3,4] 
syntax error: needs choice( ... )

Data is in black. Probabilities are in blue.
Team up and try this on the backpage first…

Pass these in and up!
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*

Bond, James Bond

*Monte Carlo* methods, *Math/CS*

Ulam, Stan Ulam
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?

N is the total number of rolls

where and how is the check for doubles being done?
Another Monty... ?
Let's make a deal...

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?

Let's play (randomly) 300 times and see!
Monte Carlo Monty Hall

```python
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 = 'Spam!'
    elif init == przDoor and sors == 'switch':  result = 'pmfp.'
    elif init != przDoor and sors == 'switch':  result = 'Spam!'
    else:                                        result = 'pmfp.'

    print 'You get the', result

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

Your initial choice!

'switch' or 'stay'

number of times to play
Let's make a deal: XKCD's take...

... what if you considered the goat the grand prize!?
If you win some SPAM... ?  or pmfp... ?
If you win some SPAM... ? or pmfp... ?
An example closer to home

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(s,nsteps)**

- take nsteps random steps starting at s

**rwsteps(s,low,hi)**

- take random steps starting at s until you reach either low or hi
An overworked 5C student (S) leaves H/S after their "late-night" breakfast–or lunch. Each moment, they randomly stumble toward class (N) or the Dorm (S). Once the student arrives at the dorm or classroom, the trip is complete. Write a program to model and analyze! this scenario...

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

\[
\text{rwpos}(s, \text{nsteps}) \quad \text{rwsteps}(s, \text{low}, \text{hi})
\]

- Take \text{nsteps} random steps starting at \( s \)
- Take random steps starting at \( s \) until you reach either \( \text{low} \) or \( \text{hi} \)
Lab!  *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...
Python's ability? It varies...
Python's ability? It varies...
import time
from turtle import *

def draw():  # define it!
    shape('turtle')
    # pause
    time.sleep(2)
    # drawing...
    width(5)
    left(90)
    forward(50)
    right(90)
    backward(50)
    down() or up()  # is the pen on the "paper"?
    color('darkgreen')
    tracer(1) or tracer(0)
    width(5)

# run it!
draw(); done()

http://docs.python.org/library/turtle.html
Turtle happiness?

some Pythons need

```python
done()
```

after turtle drawing!

This releases control of the turtle window to the computer (the operating system)
Problem: *Terminator Error*

Solution: *Just run it again!*
Single-path recursion

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

    # run
    tri()

(1) 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

(2) How about any regular N-gon?

def poly(n, N):
    """ n sides of an N-gon"""
    if n == 0: return
    else:
        forward(100)  # one side
        left(360.0/N) # turn 360/N
        poly(n-1, N)  # draw rest
Be the turtle!

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

```python
def chai(dist):
    ""
    mystery fn! ""
    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)
```

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?

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

```python
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!

one possible result of `rwalk(20)`

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?
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 )

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

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

"Single-path" (or counting) recursion
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)

Single-path recursion

What does \texttt{chai(100)} do here?

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

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

Now, what does chai(100) do?

Branching recursion
Cyriak: *conceptually disruptive* recursion...

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

\texttt{spiral(100, 90, 0.8)}

\texttt{spiral( initLength, angle, multiplier )}
lab ~ hw2pr1

fractal art

spiral(100, 90, 0.8)

spiral(80, 90, 0.8)

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)
svtree( trunkLength, levels )

svtree( 100, 5 )

svtree( 75, 4 )

levels == 5
levels == 4
levels == 3
levels == 2
levels == 1
levels == 0
(no drawing)

What steps does the turtle need to take before recursing?
svtree( trunkLength, levels )

Be sure the turtle always returns to its starting position!

step #1: go forward...

levels == 5

step #2: turn a bit...

step #3: draw a smaller svtree!

steps #4: turn to another heading

step #5: draw another smaller svtree!

step #6: get back to the start by turning and moving!
svtree( trunkLength, levels )

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…
The Koch curve

snowflake(100, 0)  snowflake(100, 1)  snowflake(100, 2)

snowflake(100, 3)  snowflake(100, 4)  snowflake(100, 5)
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!

seven-cornered confetti