Cyriak: *conceptually disruptive* recursion...

### CS 5 Today

**hw2** due Mon. 9/23

Lots of tutoring...

---

**Fractals and Turtles**

Baaa

---

**More Eyes!**

CS 5 alien on strike!

---

CS 5 green mascot representing today’s terrestrial theme

---

How random!
Cyriak: *conceptually disruptive recursion...*

Fractals and Turtles

**Applications!**

hw2 due Mon. 9/23

Lots of tutoring...

How random!
\[ \text{dot}([3,2,4],[4,7,4]) = 3*4 + \text{dot}([2,4],[7,4]) + 2*7 + \text{dot}([4],[4]) + 4*4 \]

Sequential design...
\textbf{Recursive design...}
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([], [])
16.0

30.0
42.0

L = [3,2,4] and K = [4,7,4]
L = [2,4] and K = [7,4]
L = [] and K = []

slow and steady!
There are four different values of $L$ and four different values of $K$ – all alive, simultaneously, in the stack.
Recursion's idea:

You handle the FIRST
Recursion handles the REST
Recursion's idea:

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

You handle the FIRST
Recursion handles the REST

You handle the FIRST
Recursion handles the REST

**first**

**rest**

handle the FIRST of L
handle the FIRST of K

handle the REST of L
handle the REST of K

**combine**

**recurse w/the rest**

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

You handle the FIRST
Recursion handles the REST

---

You handle the FIRST
Recursion handles the REST

- You handle the FIRST of L
- You handle the FIRST of K

handle the first

handle the FIRST of L
handle the FIRST of K

first

handle the REST of L
handle the REST of K

recurse w/the rest

rest
Some *random* asides...

```python
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 *

choices 1 element from the sequence L
choice( L )

... or 1 character from a string
choice('mudd')

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

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

```python
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))  →  [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`

```python
>>> 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:
        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

```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 )
```

code available in hw2pr2
from random import *
choice( [1,2,3,2] )
choice( list(range(5))+[4,2,4] )
choice( list(range(7)) )
choice( '1,2,3,4' )
choice( ['1,2,3,4'] )
choice( '[1,2,3,4]' )
uniform( -20.5, 0.5 )
choice(0,1,2,3,4)
choice([list(range(5))])
choice[[list(range(5))]]

What's the most likely return value here?
What's the most likely return value here?
What's the most likely return value here?
What's the most likely return value here?

More likely even or odd? 0 is even!

What are the chances of this being > 0?

Which two of these 3 are syntax errors?
Also, what does the third one – the one syntactically correct – actually do?
from random import *
choice([1,2,3,2])

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

choice(list(range(7)))

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

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

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

uniform(-20.5, 0.5)

choice(0,1,2,3,4)
choice([list(range(5))])
choice[list(range(5))]
from random import *
choice([1,2,3,2])

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

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

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

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

choice[ list(range(5)]

choice( list(range(7)]

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

What's the most likely return value here? 2 2/4 probabilities in blue...

Data is in black. Probabilities are in blue.

Pass these eastward!

careful on these...

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

1/1 chance
The two *Monte Carlos* and their denizens...

Monte Carlo casino, *Monaco*

Monte Carlo methods, *Math/CS*

Insights via *random trials*
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

```python
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
```

*where and how* is the check for doubles being done?
Monte Carlo *Let's Make a Deal*...
Monte Carlo  Let's Make a Deal...
Monte Carlo  *Let's Make a Deal...*

Inspiring the *Monty Hall paradox*
Let's make a deal: XKCD's take...

Monty Hall

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

...and my yard has so much grass, and I'll teach you tricks, and...

...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

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

we made a sale!!

Phoebe via cs.hmc.edu
to dodds

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

Phoebe
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)
rwsteps(st, low, hi)
```

- `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

An overworked 5C student \( S \) leaves H/S after their "late-night" breakfast. 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. Write a program to model and analyze! this scenario...

\[
\text{Your task: To create this as an "ASCII" animation}
\]

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

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

No way this is real… but it is!

www.gvetchedinetime.com
more *usual* etch-a-sketch work...
**Single-path recursion**

```python
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:

```python
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...

```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)
```

```python
In [33]: poly_setup()
In [36]: poly(9,9)
In [37]: 
```
Turtle's ability?  It varies widely!
Warning: Terminator error!

Problem: Terminator Error

Solution: Just run it again!
**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)
    # recurse here?
    right(90)
    forward(dist)
    left(90)
    # recurse here?
    left(90)
    forward(dist/2.0)
    right(90)
    backward(dist)
```

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

(2) 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?
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
**Single-path recursion**

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

```python
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 \texttt{chai(100)} do?

```python
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.
lab ~ hw2pr1

fractal art

$\text{spiral}(100, 90, 0.8)$

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

fractal art

spiral(100, 90, 0.8)
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 )

Be sure the turtle always returns to its starting position!

Branching recursion!
svtree( trunkLength, levels )

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)

*Single-path* or *Branching* recursion here?
Recursive art? Create your own...

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