Cyriak: *conceptually disruptive* recursion...
Welcome to IST338... Be sure to watch your head!
CS 5: *now recursing...*

We're computationally complete!

Or *re-cursing*, depending on your feelings about recursion!

**Hw 2 – due Friday 2/20 ~ as usual**

pr0 reading – *Watson!*
pr1 lab – Turtle!
pr2 – Monte Carlo simulation
pr3+4 – extra-credit probs!

What's next?

putting Python *to work!*

& adding *building-blocks*
Recursive design works...

```python
def mylen(s):
    """ input: any string, s
    output: the number of characters in s
    """
    if s == '':
        return 0
    else:
        return 1 + mylen(s[1:])
```

... by solving a smaller version of the same problem!
Behind the curtain: how recursion works...

```python
def mylen(s):
    if s == '':
        return 0
    else:
        return 1 + mylen(s[1:])
```

```python
mylen('cs5')
```

1 + mylen('s5')

1 + 1 + mylen('5')

1 + 1 + 1 + mylen('')

1 + 1 + 1 + 0
```python
def mymax(L):
    if len(L) == 1:
        return L[0]
    elif L[0] < L[1]:
        return mymax(L[1:])
    else:
        return mymax(L[0:1] + L[2:])
```

```
mymax( [1, 4, 3, 42, -100, 7] )
mymax( [4, 3, 42, -100, 7] )
mymax( [4, 42, -100, 7] )
mymax( [42, -100, 7] )
mymax( [42, 7] )
mymax( [42] )
42
```
def power(b, p):
    """ returns b to the p power
    Use recursion, not **
    Inputs: int b, int p:
    the base and the power
    """

2^0 == 1
power(2, 0) -> 1

2^5 == 2 * 2^4
power(2, 5) -> 2 * 2^4

2^p == 2 * 2
power(2, p) -> 2 * power(..., ...)

power(b, p) ->
def power(b, p):
    """ returns b to the p power
    Use recursion, not **
    Inputs: int b, int p:
    the base and the power
    """
    if p == 0:
        return 1
    else:
        return 2 * power(b, p - 1)

Want more power?
Handle negative p values w/elif.
E.g., power(5, -1) == 0.2
power( 2, 5 )

2 * power( 2, 4 )

2 * 2 * power( 2, 3 )

2 * 2 * 2 * power( 2, 2 )

2 * 2 * 2 * 2 * power( 2, 1 )

2 * 2 * 2 * 2 * 2 * power( 2, 0 )

2 * 2 * 2 * 2 * 2 * 1

32
def sajak(s):
    """ returns the number of vowels in the input string, s """

    sajak('') → 0

    sajak('okay') → 1 + sajak('ay')

    sajak('what') → 0 + sajak('at')
def sajak(s):
    """ returns the number of vowels in the input string, s """
    if s == '':
        return
    elif else:
        return

    Base case test

    Want more Pat?

    What 7-letter English word w maximizes sajak(w)?

    Try it!
sajak( 'eerier' )
1+ sajak( 'erier' )
1+ 1+ sajak( 'rier' )
1+ 1+ 0+ sajak( 'ier' )
1+ 1+ 0+ 1+ sajak( 'er' )
1+ 1+ 0+ 1+ 1+ sajak( 'r' )
1+ 1+ 0+ 1+ 1+ 0+ sajak( '' )
1+ 1+ 0+ 1+ 1+ 0+ 0

4
def power(b, p):
    """ inputs: base b and power p (an int)
    implements: b**p
    """
    if p == 0:
        return 1.0
    elif p < 0:
        return ____________________
    else:
        return b * power(b, p-1)
sajak(s): \# of vowels in s

Base case? When there are no letters, there are ZERO vowels

Rec. step? Look at the initial character.

if s[0] is NOT a vowel, the answer is

\[ \text{sajak}( s[1:] ) \]

if s[0] is a vowel, the answer is

\[ 1 + \text{sajak}( s[1:] ) \]
def sajak(s):
    if s == '':
        return 0
    elif s[0]== 'a' or s[0]== 'e' or...

    Is s[0] a vowel?

    but how to check for vowels?
Python is...  

>>> 'i' in 'team'
False

>>> 'cs' in 'physics'
True

>>> 'i' in 'alien'
True

>>> 42 in [41, 42, 43]
True

>>> 3*'i' in 'alien'
False

>>> 42 in [[42], '42']
False
def sajak(s):
    if len(s) == 0:
        return 0
    elif s[0] in 'aeiou':
        return 1 + sajak(s[1:])
    else:
        return 0 + sajak(s[1:])

let's input 'eerier' for s
The key to understanding recursion is, first, to understand recursion.

- a former student

Good luck with Homework #1

tutors @ LAC all week!

It's the eeriest!
import random
from random import *

The choice function chooses 1 element from the sequence L.

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

range(1,5) → [1,2,3,4]

How could you get a random int from 0 to 99 inclusive?

uniform(low, hi)

Aargh – so close!
from random import *

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

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

print the guesses ?
slow down...
return the number of guesses ?
investigate expected # of guesses?!!?
Empirical Hypothesis Testing...

*a.k.a.*

*How many guesses do we expect in order to find the correct number?*
Recursive guess-counting

```python
from random import *
import time

def guess( hidden ):
    """ guessing game ""
    compguess = choice( 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 )
```
from random import *

choice([1,2,3,2]) —— What are the chances this returns a 2?

choice(range(1,5)+[4,2,4,2]) —— What are the chances of this returning a 4?

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

choice([range(5)])

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

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

choice('1,2,3,4]

choice([range(5)])

uniform(-20.5, 0.5) —— What're the chances of this being > 0?
Data is in black. Probabilities are in blue.

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

choice(range(1,5)+[4,2,4,2 ])  # What are the chances of this returning a 4?
3/8

choice(0,1,2,3,4 )  # syntax error

choice( range(5) )  # syntax error

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

choice('1,2,3,4' )  # What’s the most likely return value here? '
1,2,3,4'
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? '
1,2,3,4'
3/9

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

uniform(-20.5, 0.5)  # What’re the chances of this being > 0? 1/42
```
The two *Monte Carlo* 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*

Stanislaw Ulam (Los Alamos badge)

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( range(1,7) )
        \{ two dice from 1-6 inclusive \}
        if d1 != d2:
            return 0+countDoubles( N-1 ) # don't count it
        else:
            return 1+countDoubles( N-1 ) # COUNT IT!
```

*where and how is the check for doubles?*
Empirical Hypothesis Testing...

*How many guesses do we expect in order to find the correct number?*
Empirical Hypothesis Testing...

# this line runs guess(42) 1000 times!
LC = [ guess(42) for x in range(1000) ]

# Let's look at the first 10 of them:
print LC[0:10]

# Let's find the average:
print "av. guesses: ", sum(LC)*1.0/len(LC)

Run it a zillion times!

Hah! Now I see why they told me I'd be making a zillion euros!
Empirical Hypothesis Testing...

a.k.a.

How likely are we to roll doubles on two six-sided dice?

Hah! Now I see why they told me I'd be making a zillion euros!
# this runs the doubles-counter 600 times...
cd( 600 )

# Run _that_ 100 times (60,000 rolls total!)
DBLS = [ cd(600) for x in range(100) ]

# Look at the first 10 of these
print DBLS[0:10]

# Let's find the average:
print "av. dbls/600:", sum(DBLS)*1.0/len(DBLS)

needed a less continental name...
On balance?

or maybe lighter is better?
Data Functions

\[ \text{sq}(x) \text{ for } x \text{ in } [8, 9, 10] \]

...together
List Comprehensions

```python
>>> [2*x for x in [0, 1, 2, 3, 4, 5]]
[0, 2, 4, 6, 8, 10]
```

What's the syntax saying here?
List Comprehensions

The same as `map`, only better!

this "runner" variable can have *any* name...

```
>>> [ 2*x for x in [0,1,2,3,4,5] ]
```

and $2 \times x$ is output for each one

```
[0, 2, 4, 6, 8, 10]
```

output
List Comprehensions

```python
>>> [ 10*x for x in [0,1,2,3,4,5] if x%2==0]
result

>>> [ y*21 for y in range(0,3) ]
result

>>> [ s[1] for s in ["hi", "5Cs!"] ]
result

OK – got it. But what about that name?
List Comprehensions?

Is this really the best name Guido Van Rossum could think of?

Guido van Rossum

From Wikipedia, the free encyclopedia

Guido van Rossum (born 31 January 1956) is a Dutch computer programmer who is best known as the author of the Python programming language. In the Python community, Van Rossum is known as a "Benevolent Dictator For Life" (BDFL), meaning that he continues to oversee the Python development process, making decisions where necessary. He is currently employed by Google, where he spends half his time developing the Python language.
List Comprehensions?

```python
>>> [ 2*x for x in [0,1,2,3,4,5] ]
[0, 2, 4, 6, 8, 10]
```

Google maps?

Datafuncs?

FunLists!

A list comprehension by *any* other name would be as sweet...

---

Guido van Rossum (born 31 January 1956) is a Dutch computer programmer who is best known as the author of the Python programming language. In the Python community, Van Rossum is known as a “Benevolent Dictator For Life” (BDFL), meaning that he continues to oversee the Python development process, making decisions where necessary. He is currently employed by Google, where he spends half his time developing the Python language.
Quiz! A range of list comprehensions...
Write Python's result for each L.C.:  

\[
\begin{align*}
\text{[ } n**2 & \text{ for } n \text{ in range}(0,5) \text{ ] } \\
\text{[ } 42 & \text{ for } z \text{ in } [0,1,2] \text{ ] } & \text{[ } z & \text{ for } z \text{ in } [0,1,2] \text{ ] } \\
\text{[ } s[1::2] & \text{ for } s \text{ in } ['aces','451!'] \text{ ] } \\
\text{[ } -7*b & \text{ for } b \text{ in range}(-6,6) \text{ if } \text{abs}(b)>4 \text{ ] } \\
\text{[ } a*(a-1) & \text{ for } a \text{ in range}(8) \text{ if } a%2==1 \text{ ] }
\end{align*}
\]
Syntax ?!

```python
>>> [ 2*x for x in [0,1,2,3,4,5] ]
[0, 2, 4, 6, 8, 10]
```

at first...

a jumble of characters and random other stuff

a (frustrated!) rendering of an unfamiliar math problem
Syntax ?!

```python
>>> [ 2*x for x in [0,1,2,3,4,5] ]
[0, 2, 4, 6, 8, 10]
```

at first...

a jumble of characters and random other stuff

a (frustrated!) rendering of an unfamiliar math problem
Syntax ~ *is* CS's key resource!

a (frustrated!) rendering of an unfamiliar math problem which was likely similar to these...

Where'd the change happen?
Syntax vs. Semantics

Plus – you might consider coming to the Career Fair this Friday at HMC's LAC...
Another Monte Carlo Monty... ?
Let's make a deal...

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

... but 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?

Run it (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 = '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 )
```
CS for Insight!

How often do we win if we **SWITCH** vs. **STAY**?

*a.k.a.*

*I hope the prize isn't in Euros!*
# this runs the game once (staying...)
MCMHOnce1( 3, 'stay' )

# Run _that_ 3000 times
PRIZES = [ MCMH1(3,'stay') for x in range(3000) ]

# Look at the first 10 of these
print PRIZES[0:10]

# Let's find the total number of wins:
how to do this... ?!
An example *closer to home* ...

An overworked CGU student (S) leaves Harg. after their "late-night" breakfast. Each moment, they randomly stumble toward class (W) or their Apartment (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...

```plaintext
rwpos(s, nsteps)  rwst莫斯(s, low, hi)
```

- `rwpos(s, nsteps)` takes `nsteps` random steps starting at `s`.
- `rwst莫斯(s, low, hi)` takes random steps starting at `s` until you reach either `low` or `hi`.
An example closer to home

An overworked CGU student (S) leaves Harg. after their "late-night" breakfast. Each moment, they randomly stumble toward class (W) or their Apartment (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`
Recursion's *challenge*?

You need to see BOTH the *self-similar pieces* AND the *whole thing* simultaneously!

Nature prefers recursion, too!
Yes... and no.

Are these rules for real?
Dragon's-blood Tree
There still has to be a base case...
or else!
Cyriak: *conceptually disruptive* recursion...

is the *branching*, not the *single-path* variety.
import time
from turtle import *

def draw():
    shape('turtle')
    # pause
    time.sleep(2)
    # drawing...
    width(5)
    left(90)
    forward(50)
    right(90)
    backward(50)
    down() or up()
    color('darkgreen')
    tracer(1) or tracer(0)
    width(5)

http://docs.python.org/library/turtle.html
Single-path recursion

(1) 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
```

(2) How about any regular N-gon?

```python
def poly(n, N):
    """ draws a triangle ""
    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 does `chai(100)` draw?

```python
def chai(size):
    """ mystery! ""
    forward(size)
    left(90)
    forward(size/2.0)
    right(90)
    right(90)
    forward(size)
    left(90)
    left(90)
    forward(size/2.0)
    right(90)
    backward(size)
```

Extra! What if you called `chai(size/2)` between the `right(90)` & `left(90)` calls?

2. Finish `rwalk` so it draws a "stock-market" path: 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(45)
        forward(10)
```

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

-one possible result of `rwalk(20)`-
def chai(size):
    """ mystery! ""
    if size<9:
        return

    forward(size)
    left(90)
    forward(size/2.0)
    right(90)
    right(90)
    forward(size)
    left(90)
    left(90)
    forward(size/2.0)
    right(90)
    backward(size)

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

(2) rwalk is a random stock market walk...

Extra: Creating a bull (or a bear) market?
fractal art!

spiral(100, 90, 0.9)
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 )

What steps does the turtle need to take before recursing?

- levels == 5
- levels == 4
- levels == 3
- levels == 2
- levels == 1
- levels == 0
  (no drawing)
svtree\( (\text{trunkLength, levels}) \)

Be sure the turtle always returns to its starting position!

step #1: go forward...

levels == 5

step #2: turn a bit...

levels == 4

step #3: draw a smaller svtree!

levels == 3

step #4: turn to another heading

levels == 2

step #5: draw another smaller svtree!

levels == 1

levels == 0 (no drawing)

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

levels == 5

levels == 4

levels == 3

levels == 2

levels == 1

levels == 0

(no drawing)
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...* hw2pr4

Happy turtling!

seven-cornered confetti

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