Cyriak: conceptually disruptive recursion...
Welcome back to CS 5!

Homework 2

Problem 0: Reading + response...

Problem 1: Recursion! Do these during lab... due today (Mon. ~ lab)

Problem 2: PythonBat functions! In lab or beyond... due Sunday evening

Average of these two?
Welcome back to CS 5!

Homework 3

Problem 0: Reading + response...  
Problem 1: They're named 3+4...  due Sunday evening

Problem 2: Sleepwalking simulation!  due Sunday evening

Problem 3+4: Picobot Lab...  due Wed. ~ lab

I see the resemblance
this week's hw2pr0

**Category:** U.S. Cities.  **Clue:** Its largest airport is named for a World War II hero, its second largest for a World War II battle.

Watson
Are we in a simulation right now?
Recursion + Conditionals: *All we need!*

If we *were* in a simulation, how would *you* author it?

Lyra??
Recursion + Conditionals: *All we need!*

What's Next?

Applications!

First, let's look back...

*If we were in a simulation, how would you author it?*
\[ \text{dot}([3, 2, 4], [4, 7, 4]) \]

Sequential design...
dot([3,2,4],[4,7,4])

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

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.

Seeing the "stack" ...
Recursion's style:

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

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

**You handle the FIRST**

**Recursion handles the REST**
Recursion's style:

```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

**Combine**

- handle the FIRST of L
- handle the FIRST of K
- handle the REST of L
- handle the REST of K

**Recursion**

- handle the first
- recurse w/the rest
Some *random* asides...
Today's chat question:

*Choose your Avatar (character)*

*talk about random...!*
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 *

# 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'])
```
Some *random* asides...

```python
import random
from random import *
```

allows use of `dir(random)` and `help(random)`

all random functions are now available!

```python
choice( L )
```

chooses 1 element from the sequence L

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

... or 1 character from a string

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

```python
list(range(5))  →  [0,1,2,3,4]
```

```python
list(range(1,5))  →  [1,2,3,4]
```

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

```python
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!*
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)
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
Random "Quiz"

**from random import ***

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

What's the most likely return value here?

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

What's the most likely return value there?

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

More likely even or odd?

```
uniform(-20.5, 0.5)
```

What are the chances of this being > 0?

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

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])

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))]
Data is in black. Probabilities are in blue.

```python
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))]
```

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

What’s the most likely return value here? 2

What’s the most likely return value? 4

More likely even or odd? 0 is even!

What’s the most likely return value here? '

What’s the most likely return value here? '1,2,3,4'

What’s the most likely return value here? '

What are the chances of this being > 0? 1/42

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

*Insights via random trials*

Monte Carlo methods, Math/CS
The two *Monte Carlos*

Monte Carlo casino, *Monaco*

and their denizens...

**Bond, James Bond**

**Ulam, Stan Ulam**

**Monte Carlo methods, Math/CS**
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...*

Let's Make a Deal...
Monte Carlo  *Let's Make a Deal*...
Let's make a deal: XKCD's take...

Monty Hall

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

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

'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, 'you', result
    if result == 'Win!': return 1 + MCMH( init, sors, N-1 )
    else:
        return 0 + MCMH( init, sors, N-1 )
Monte Carlo Monty Hall
Monte Carlo Monty Hall
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

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)

take nsteps random steps starting at st

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

Your task: Create your own Monte-Carlo animation

**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 \(hi\)
The *challenge* of programming...
The *challenge* of programming...

This is somehow familiar...?!

- **syntax**
  - How it looks

- **semantics**
  - What it does

- **intent**
  - What it should do

- **human-typed input**

- **machine-produced output**

- **human-desired output**

- ?
Another language!

Let's *not only* add another language...

... *but also make it half the hw*!

*Even with three eyes, I must be misreading this!*
Another language *already*?

**Python**

*General-purpose language*

you might see 50% by the end of the term

even then, <1% of its libraries!

**Picobot**

*Special-purpose language*

you'll see 100% in the next 10 minutes

The Picobot simulator

[www.cs.hmc.edu/picobot](http://www.cs.hmc.edu/picobot)

**Picobot**

Be sure to hit "Enter rules" after making changes.
HW problems 3 and 4: Picobot!

**Goal:** full-room coverage with only *local sensing*...

Inspiration?
HW problems 3 and 4: Picobot!

**Goal:** full-room coverage with only *local sensing*...

The Roomba!
- can't tell "vacuumed" from "unvacuumed" area

Let's see it!
Surroundings

Picobot can only sense things directly to the N, E, W, and S

For example, here its surroundings are

\[ N \times W \times x \]

Surroundings are always in \textit{NEWS} order.
What are these surroundings?

Surroundings are always in NEWS order.

Wow - this one is disgusting!
Surroundings

How many distinct surroundings are there?
Surroundings

How many distinct surroundings are there?

$2^4 = 16$ possible

Aargh!
State

Picobot's memory is a single number, called its state.

State is the internal context of a computation, i.e., its subtask.

Picobot always starts in state 0.

State and surroundings represent everything Picobot knows about the world.
Picobot programming ~ *rules*

<table>
<thead>
<tr>
<th>current state</th>
<th>surroundings</th>
<th>direction</th>
<th>new state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Nxxx</td>
<td>S</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>xxxx</td>
<td>N</td>
<td>0</td>
</tr>
</tbody>
</table>

These two rules are a complete Picobot program.

**Notes**

Picobot checks its rules from the top each time. *When it finds a matching rule*, that rule runs.

```
| Start  | Step 1 | Step 2 | Step 3 | Step 4 | ...
|--------|--------|--------|--------|--------|--------
|        |        |        |        |        | ?      |
```
Picobot programming ~ rules

<table>
<thead>
<tr>
<th>current state</th>
<th>surroundings</th>
<th>direction</th>
<th>new state</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule (A)</td>
<td>0</td>
<td>Nxxx</td>
<td>S</td>
</tr>
<tr>
<td>rule (B)</td>
<td>0</td>
<td>xxxx</td>
<td>N</td>
</tr>
</tbody>
</table>

These two rules are a complete Picobot program.

Notes

Picobot checks its rules from the top each time. *When it finds a matching rule*, that rule runs.

Start

Step 1

Step 2

Step 3

Step 4

...
Picobot programming ~ *rules*

<table>
<thead>
<tr>
<th>current state</th>
<th>surroundings</th>
<th>direction</th>
<th>new state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Nxxx</td>
<td>S</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>xxxx</td>
<td>N</td>
<td>0</td>
</tr>
</tbody>
</table>

*Notes*

Picobot checks its rules from the top each time. 
*When it finds a matching rule*, that rule runs.

These two rules are a complete Picobot program.

---

**Step 1**

- Current state: 0
- Surroundings: Nxxx
- Direction: S
- New state: 0

**Step 2**

- Current state: xxxx
- Surroundings: 
- Direction: N
- New state: 0

---

...
Picobot programming ~ rules

<table>
<thead>
<tr>
<th>current state</th>
<th>surroundings</th>
<th>direction</th>
<th>new state</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule (A)</td>
<td>0</td>
<td>Nxxxx</td>
<td>S</td>
</tr>
<tr>
<td>rule (B)</td>
<td>0</td>
<td>xxxxx</td>
<td>N</td>
</tr>
</tbody>
</table>

these two rules are a complete Picobot program

Notes

Picobot checks its rules from the top each time.
When it finds a matching rule, that rule runs.
Picobot programming ~ *rules*

<table>
<thead>
<tr>
<th>current state</th>
<th>surroundings</th>
<th>direction</th>
<th>new state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Nxxx</td>
<td>S</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>xxxx</td>
<td>N</td>
<td>0</td>
</tr>
</tbody>
</table>

*Notes*

Picobot checks its rules from the top each time. *When it finds a matching rule*, that rule runs.

These two rules are a complete Picobot program.
**Picobot programming ~ rules**

<table>
<thead>
<tr>
<th>current state</th>
<th>surroundings</th>
<th>direction</th>
<th>new state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Nxxx</td>
<td>S</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>xxxx</td>
<td>N</td>
<td>0</td>
</tr>
</tbody>
</table>

These two rules are a complete Picobot program.

**Notes**

Picobot checks its rules from the top each time. *When it finds a matching rule*, that rule runs.

---

**Start**

- **Step 1**
- **Step 2**
- **Step 3**
- **Step 4**

*These cycle back and forth...*
Picobot acts through a set of rules.

Each rule expresses your intent for Picobot!

### Rules

- **I am in state 0.**
- **My surroundings are xxWS.**

If Picobot's in state 0 seeing xxWS,

Then move North, and "change" to state 0.
Wildcards

Asterisks * are wild cards. They match walls *or* empty space:

N must be empty

I only care about NORTH being EMPTY

current state  

surroundings  

direction  

new state  

EWS may be wall *or* empty space

8 surroundings in one rule

that's it!
1. Run Picobot! Which rule A, B, or C runs first? ______
   1a. How many times does rule (A) run? ______
   1b. How many times does rule (B) run? ______
   1c. How many times does rule (C) run? ______

2. Picobot stops when no rule matches. **Where does it stop?**

3. Add a rule so that Picobot continues **back upward!**

**Extra #1** Rule A has a bug! What is it?
**Extra #2** Add rules to finish exploring the empty room from any starting point...
**Extra #3** **How to do this in only 6 rules total?!**
1. Run Picobot! Which rule A, B, or C runs first? _______
   1a. How many times does rule (A) run? _______
   1b. How many times does rule (B) run? _______
   1c. How many times does rule (C) run? _______

2. Picobot stops when no rule matches. Where does it stop?

3. Add a rule so that Picobot continues back upward!

Extra #1: Rule A has a bug! What is it?
Extra #2: Add rules to finish exploring the empty room from any starting point...
Extra #3: How to do this in only 6 rules total?!
Warning!  What's wrong here?

<table>
<thead>
<tr>
<th>state</th>
<th>surroundings</th>
<th>direction</th>
<th>new state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td><em>x</em>**</td>
<td>S</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>***x</td>
<td>N</td>
<td>0</td>
</tr>
</tbody>
</table>

Note: Picobot checks its rules from the top each time. *When it finds a matching rule*, that rule runs.

*These two rules are a broken Picobot program!*
Warning! *What's wrong here?*

<table>
<thead>
<tr>
<th>state</th>
<th>surroundings</th>
<th>direction</th>
<th>new state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x***</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>***x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

These two situations **COULD BE** the same!

Picobot checks its rules from the top each time. *When it finds a matching rule*, that rule runs.

Notes

There can only be **ONE** rule per situation! and a "situation" is **state** and **surroundings**.
CS ~ Complexity Science

Problem 3
Shortest Picobot program:

6 rules

Problem 4
Shortest Picobot program:

8 rules

Problem 5 (extra!)

Problem 6 (extra!)
Maze strategies?
Maze solving algorithm

There are a number of different maze solving algorithms, that is, automated methods for the solving of mazes. The random mouse, wall follower, Pledge, and Trémaux algorithms are designed to be used inside the maze by a traveler with no prior knowledge of the maze, whereas the dead-end filling and shortest path algorithms are designed to be used by a person or computer program that can see the whole maze at once.

Mazes containing no loops are known as "standard", or "perfect" mazes, and are equivalent to a tree in graph theory. Thus many maze solving algorithms are closely related to graph theory. Intuitively, if one pulled and stretched out the paths in the maze in the proper way, the result could be made to resemble a tree.\[^1\]
Maze strategies?

**Right Hand Rule**

Keep your "right hand" on the wall, Picobot!

Why might this be difficult for Picobot?
Maze strategies?

Keep your "right hand" on the wall, Picobot!

Right Hand Rule

State 0
State 1
State 2
State 3

We'll need to use state to represent the direction Picobot is facing.
Suppose Picobot wants to traverse a maze *with its right hand always on the wall*...

(A) CORRIDOR rule

*If you're facing N with a wall at right and space ahead then go forward”*

(B) INTERSECTION rule

“If you're facing North and lose the wall, then get over to the wall now!“

(C) DEAD END rule

Write 1 or 2 rules to tell Picobot to do the right thing if it hits a dead end.

Repeat this IDEA for all four states, representing all four *facing directions.*
Suppose Picobot wants to traverse a maze *with its right hand always on the wall*...

**(A) CORRIDOR rule**

*If you're facing N with a wall at right and space ahead, then go forward*.

\[
\begin{array}{ccc}
0 & xE** & -> & N & 0 \\
\end{array}
\]

**state 0 means "still facing north"**

**(B) INTERSECTION rule**

*If you're facing North and lose the wall, then get over to the wall now!*

\[
\begin{array}{ccc}
0 & **x** & -> & E & 1 \\
\end{array}
\]

**state 1 means "now facing east"**

**(C) DEAD END rule**

Write 1 or 2 rules to tell Picobot to do the right thing if it hits a dead end.

\[
\begin{array}{ccc}
0 & NE** & -> & X & 2 \\
\end{array}
\]

**state 2 means "now facing west"**

Repeat this IDEA for all four states, representing all four *facing directions*.
Hooray!?!
Lab/hw

Malia + Deanna!
Piazza + questions!

Happy Picobotting!

You are not alone!

And, good luck with *animating* Python!