They call that an alien?

Spock *mind-melds* three-eyed aliens!

Provably.

http://www.youtube.com/watch?v=iapcKVn2DqY
http://www.youtube.com/watch?v=yuEZEyDdmvQ
Getting started with RPS...

```python
# import random

print("Welcome to rock/paper/scissors, human!\n")

comp = random.choice(['rock', 'paper', 'scissors'])
user = input(" Choose wisely: ")

print(" You chose", user)
print(" I chose", comp)

if user == 'rock':
    if comp == 'paper':
        print(" paper defeats rock - I win!")
```
Lab lookback...

Lab's goal: Get things working
Complete most of the HW problem

print "Thirty Three is", sqrt(4)/.4 + factorial(4) + 4
print "Victory!"

Four fours is ~
sometimes too many...
othertimes too few...

and never enough!
-- Prof. Su
Help? via email/Piazza...

for some questions, Piazza is a great resource:

this link:

this Q&A page
Next up for CS 5 ...

Homework 0

due Sun. night (11:59pm)

Problem 0: Reading + response...
Problem 1: Four-fours program: Can be done for lab...
Problem 2: Rock-paper-scissors program (*Maybe* done already!)
Problem 10: Adventure with Python!
Problems 3-4: Picobot! empty room (3) maze (4)
Next up for CS 5!

Homework 0

Problem 0: Reading + response...

Problem 1: Four-fours program: Can be done for lab...

Problem 2: Rock-paper-scissors program (Maybe done already!)

Problem 10: Adventure with Python!

Problems 3-4: Picobot! empty room (3) maze (4)
Problem 0?

Typically an article on CS or an application...

Submit a one-paragraph response

Small part (5 pts)

- 5 – insightful, careful
- 4 – thoughtful
- 3 – complete, on topic
- 0-2 – less than complete

A few sentences that raise or address questions, using the article as a guide.

This week's article might not seem like CS at first...
Seventy years ago, in 1940, a popular science magazine published a short article that set in motion one of the trendiest intellectual fads of the 20th century. At first glance, there seemed little about the article to augur its subsequent celebrity. Neither the title, “Science and Linguistics,” nor the magazine, M.I.T.’s Technology Review, was most people’s idea of glamour. And the author, a chemical engineer who worked for an insurance company and moonlighted as an anthropology lecturer at Yale University, was an unlikely candidate for international superstardom. And yet Benjamin Lee Whorf let loose an alluring idea about language’s power over the mind, and his stirring prose seduced a whole generation into believing that our mother tongue restricts what we are able to think.
But then a remote Australian aboriginal tongue, Guugu Yimithirr, from north Queensland, turned up, and with it came the astounding realization that not all languages conform to what we have always taken as simply “natural.” In fact, Guugu Yimithirr doesn’t make any use of egocentric coordinates at all. The anthropologist John Haviland and later the linguist Stephen Levinson have shown that Guugu Yimithirr does not use words like “left” or “right,” “in front of” or “behind,” to describe the position of objects. Whenever we would use the egocentric system, the Guugu Yimithirr rely on cardinal directions. If they want you to move over on the car seat to make room, they’ll say “move a bit to the east.” To tell you where exactly they left something in your house, they’ll say, “I left it on the southern edge of the western table.” Or they would warn you to “look out for that big ant just north of your foot.” Even when shown a film on television, they gave descriptions of it based on the orientation of the screen. If the television was facing north, and a man on the screen was approaching, they said that he was “coming northward.”
Last time...

What is CS?

CS is the study of complexity

How can it be done?

How well can it be done?

Can it be done at all?

CS's 6 big questions are here.

But only one is programming.
Do you see which?

Can you solve this problem?

Can you create a process to solve such problems?

How quickly can you find solutions?

Do you have the “best” solution?

Is every problem solvable?

Is there a way to tell?
There isn’t always!

CS != Programming
What *is* programming?

Programming as recipe-writing

vs.

Programming as learning a foreign language

1) Expect it to be different!

2) Don't memorize anything!

3) Immerse == Experiment!

Baggage!
What about the Python programming language?
Python?

One possible relationship...
Python!

One possible relationship...

Happy co-existence...
*It can even be comfy!*
The *foreign language* of python...

- **Syntax**: How it looks
- **Semantics**: What it does
- **Intent**: What it should do
The **foreign language** of python...

**Syntax**
- How it looks

**Semantics**
- What it does

**Intent**
- What it should do

```python
name = raw_input('Hi... what is your name? ')  
print  
# prints 

if name == 'Eliot' or name == 'Ran':  
    print 'I''m "offline." Try later.'  

elif name == 'Zach':  
    # is it Zach?  
    print 'Zach Quinto...?  
    # is it Zach?  
    print 'No?  
    # in all other cases...  
    print 'Welcome', name, '!'  

else:  
    # in all other cases...  
    print 'Welcome', name, '!'  
    my_choice = random.choice( ['R', 'P', 'S'] )  
    print 'My favorite object is', my_choice, '!'  
```

This program should greet its user appropriately.
The *foreign language* of python...

**syntax**  
How it looks

**semantics**  
What it does

**intent**  
What it should do

```python
name = raw_input('Hi... what is your name? ')  
print  

if name == 'Eliot' or name == 'Ran':  
    print 'I''m "offline." Try later.'

elif name == 'Zach':  
    # is it Zach?  
    print 'Zach Quinto...? No? Oh.'

else:  
    # in all other cases...  
    print 'Welcome', name, '!'  

my_choice = random.choice( [ 'R', 'P', 'S' ] )  
print 'My favorite object is', my_choice, '!'  
```

*This program should greet its user appropriately.*

*human-desired output*
The *foreign language* of python...

**syntax**
How it looks

**semantics**
What it does

**intent**
What it should do

```python
name = raw_input('Hi... what is your name? ')
print  # prints a blank line

if name == 'Eliot' or name == 'Ran':
    print 'I\'m "offline." Try later.'

elif name == 'Zach':
    # is it Zach?
    print 'Zach Quinto...?', 'No?','Oh.'

else:  # in all other cases...
    print 'Welcome', name, '!!'
    my_choice = random.choice( [ 'R','P','S' ] )
    print 'My favorite object is', my_choice, "!"
```

The foreign language of python…

**syntax**

**semantics**

**intent**

How it looks

What it does

What it should do
name = raw_input('Hi... what is your name? ')  
print # prints a blank line

if name == 'Eliot' or name == 'Ran': 
    print 'I\'m "offline." Try later.'

elif name == 'Zach': # is it Zach? 
    print 'Zach Quinto...?', 'No?', 'Oh.'

else: # in all other cases...
    print 'Welcome', name, '!!'
    my_choice = random.choice( [ 'R', 'P', 'S' ] )
    print 'My favorite object is', my_choice, '!'
Flowchart...

a graphical view of a program's semantics

machine-produced output
The **foreign language** of python...

- **syntax**
  - How it looks

- **semantics**
  - What it does

- **intent**
  - What it should do

---

How Python *looks*!

- how punctuation is used
- the language **keywords** used
- use of whitespace

- peculiarities of formatting
- how behavior is affected ...
The *foreign language* of Python...

**syntax**
- How it looks

**semantics**
- What it does

**intent**
- What it should do

How Python *looks*!
- how punctuation is used
- the language **keywords** used
- use of whitespace

- peculiarities of formatting
- how behavior is affected...
The *challenge* of programming...

- **syntax**
  - How it looks

- **semantics**
  - What it does

- **intent**
  - What it should do

**human-typed input** → **machine-produced output** → **human-desired output**
import random

user = input( "Choose your weapon! " )
comp = random.choice( ['rock','paper','scissors'])
print('user (you) chose:', 'user')
print('comp (me!) chose:' comp)

if user == rock and comp == 'paper'
    print('The result is, YOU LOSE.'
    print('unless you're a CS 5 grader, then YOU WIN!')

(1) Find and correct as many errors as you can in this code:

(2) This one line does **three** things... what are they?

(3) Extra! Can you find 7 punctuation marks used in *more than one way* here?
import random

user = input( "Choose your weapon! " )
comp = random.choice( ['rock','paper','scissors'] )
print('user (you) chose:', user)
print('comp (me!) chose:' , comp)

if user == 'rock' and comp == 'paper'
    print('The result is, YOU LOSE.'
    print('unless you're a CS 5 grader, then YOU WIN!')
Syntax challenge!

import random

user = input( "Choose your weapon! " )
comp = random.choice( ['rock','paper','scissors'] )
print('user (you) chose:', user)
print('comp (me!) chose:', comp)

if user == 'rock' and comp == 'paper':
    print('The result is, YOU LOSE.')
    print('unless you\'re a CS 5 grader, then YOU WIN!')

(1) Find and correct as many errors as you can here...
(2) This line is doing three things... what are they?
(3) Punctuation used in more than one way: ( ), . ' = , :
Your Quest!? 

Create a short text-adventure in Python...

Use at least five control structures with decisions: (if/elif/else)

```
import time

def adventure():
    """ this function runs one session of interactive fiction ""
    Well, it's "fiction," depending on the pill color chosen...
    inputs: no inputs  (prompted text doesn't count as input)
    outputs: no outputs  (printing doesn't count as output)

Title for your adventure: The Quest.

Notes on how to "win" or "lose" this adventure:
To win, choose the table.
To lose, choose the door.
```

We look forward to adventuring!
Another language!

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

... *but also make it half the hw*!
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
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 x W x W x N

Surroundings are always in NEWS order.
What are these surroundings?

Surroundings are always in NEWS order.

Wow - this one is disgusting!
How many distinct surroundings are there?

\[ 2^4 = 16 \text{ possible} \]
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.

---

| Step |  
|------|---
| **Start** | ![Start](image)
| **Step 1** | ![Step 1](image)
| **Step 2** | ![Step 2](image)
| **Step 3** | ![Step 3](image)
| **Step 4** | ![Step 4](image)
| … | ![…](image)
| ? | ![?](image)
**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.
Picobot checks its rules from the top each time. 

*When it finds a matching rule*, that rule runs.

---

**Notes**

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>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.
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.

---

**Step 1**

**Step 2**

**Step 3**

**Step 4**

...
Picobot programming ~ *rules*

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.
Rules

I am in state 0. My surroundings are xxWS.

Picobot acts through a set of rules

Each rule expresses your intent for Picobot!

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:

I only care about NORTH being EMPTY

EWS may be wall or empty space

N must be empty

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?!
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>S</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>***x</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.

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

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

pr. 5 (extra!)

pr. 6 (extra!)
Maze strategies?
Maze solving algorithm

From Wikipedia, the free encyclopedia

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]

Right Hand Rule
Maze strategies?

**Right Hand Rule**

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

Why might this be **difficult** for Picobot?
Maze strategies?

Right Hand Rule

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

We'll need to use state to represent the direction Picobot is facing.

State 0
State 1
State 2
State 3
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*”

(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.*
Hooray!?!?

Is it working?
Post questions to piazza...

You are not alone!

Come to tutoring hours!
Post questions to piazza...

Lab/hw

Happy Picobotting!

And, good luck with the adventure of Python!
(A) Make sure you can submit!
log in to the submission system...

(B) Complete Picobot's empty room...
submit – and check the auto-graded score!

(Option #1) Complete the Picobot maze...
(Option #2) Complete rock-paper-scissors...
(Option #3) Python Adventure!