They call that an alien?

Spock *mind-melds* three-eyed aliens!

Provably.
Lab 1: Check!
Lab lookback...

*Lab's goal:* Get things working
Complete 25-50% of the hw

*Finished with lab?* OK! No *need* to stay longer

```
print("Nineteen is", factorial(4)-4-(4/4))
print("Nineteen is", factorial(4)-(4/4)-4)
```
Four fours is ~

*sometimes too many...*

*other times too few...*

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>(4\times\frac{4}{4})</td>
<td>(34)</td>
<td>(4! + \frac{4}{4} + 4)</td>
<td>(67)</td>
<td>(4! + \frac{4}{4} + 4)</td>
<td>(68)</td>
<td>(4! + \frac{4}{4} + 4)</td>
<td>(69)</td>
<td>(4! + \frac{4}{4} + 4)</td>
<td>(70)</td>
<td>(4! + \frac{4}{4} + 4)</td>
<td>(71)</td>
<td>(4! + \frac{4}{4} + 4)</td>
<td>(72)</td>
</tr>
</tbody>
</table>
15  \(4(A) - \frac{9}{4}\)
16  \(4(A) / \frac{9}{4}\)
17  \(4(A) + \frac{9}{4}\)
18  \((A - 4) + \frac{9}{4}\)
19  \((A - 4) + \frac{9}{4}\)
20  \((A + 9) - \frac{9}{4}\)
21  \((A - 4) + \frac{9}{4}\)
22  \((A + 9) - \frac{9}{4}\)
23  \((A - 4) + \frac{9}{4}\)
24  \((A + 9) - \frac{9}{4}\)
25  \((A + 9) - \frac{9}{4}\)
26  \((A + 9) - \frac{9}{4}\)
27  \((A + 9) - \frac{9}{4}\)
28  \((A + 9) - \frac{9}{4}\)
29  \((A + 9) - \frac{9}{4}\)
30  \((A + 9) - \frac{9}{4}\)
31  \((A + 9) - \frac{9}{4}\)
32  \((A + 9) - \frac{9}{4}\)
33  \((A + 9) - \frac{9}{4}\)
Email help:  \textit{Start w/ Piazza...}

\textit{for many questions,} Piazza \textit{is a great resource:}

\textbf{Useful/Helpful}
- GradeScope
- CS5 Piazza
- Crucital

\textbf{this link:}

\textbf{this Q&A page}
In-person help: "grutoring"

every day there are tutoring hours in the LAC lab

this link:

Useful/Helpful

- GradeScope
- CS5 Piazza
- Grutoring!

this page

<table>
<thead>
<tr>
<th>Monday in the LAC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Monday 6pm-8pm</strong></td>
</tr>
<tr>
<td>CS5 at HMC’s LAC (up to 4)</td>
</tr>
<tr>
<td>DON'T sign up</td>
</tr>
</tbody>
</table>
Tutoring location @ HMC: **LAC**

Tutoring hours are in the Linde Activities Center computer lab (LAC lab)

Come by!
My office hours are Thurs 2-4
LOTS of tutoring hours, as well...

Lots of grutors – all as friendly as Henry and Jose!

Tutoring location @ HMC: **LAC**
Welcome back to CS 5!

Homework 1

due Mon. 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!)

Problems 3-4: Picobot! empty room (3) maze (4)
Welcome back to CS 5!

Homework 1
due Mon. 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!)
- Problems 3-4: Picobot! empty room (3) maze (4)
Problem 0?

Typically an article on CS or an application...

Submit a one-paragraph response

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

Small part (5 pts)

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

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.
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. 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 Havieland 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.”

Better: https://www.youtube.com/watch?v=zl7eQbGASF0

Old: https://www.nytimes.com/video/movies/100000004824816/anatomy-of-a-scene-arrival.html @ 1:33 or so...
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?

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!)

But only one is programming. Do you see which?

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?
One possible relationship...
Happy co-existence...

It can even be comfy!

One possible relationship...
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 a blank line
if name == 'Eliot' or name == 'Ran':
    print 'I \"im "offline." Try later.'
elif name == 'Zach':
    # is it Zach?
    print 'Zach Quinto...?\n    No?\n    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.
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\n    
elif name == 'Zach':                      
    print 'Zach Quinto...?\n        , No?\n        , Oh.'
else:                                     
    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...
The *foreign language* of python...

```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, "!
```

*Syntax*
- How it looks

*Semantics*
- What it does

*Intent*
- What it should do

The foreign language of python...

How it looks  
- syntax

What it does  
- semantics

What it should do  
- intent
Flowchart...

a graphical view of a program's **semantics**

prompt and user input

name

decision

Is name equal to 'Eliot' or 'Ran'

If YES: print message for Eliot or Ran

If NO: decision

Is name equal to 'Zach'

If YES: print message for Zach

If NO: print message for everyone else
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

**Human-typed input**

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...

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

??
The *challenge* of programming...

Look deep into my eyes...

**syntax**
- How it looks

**semantics**
- What it does

**intent**
- What it should do

**human-typed input** → **machine-produced output** → **human-desired output**
```python
# # RPS example starting point

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)
print()

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

Syntax challenge!

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

```python
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!')
```

(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?
Syntax challenge!

(1) Find and correct as many errors as you can here...

```python
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! ')
```

(2) This line is doing **three** things... what are they?

- (1) prints the "weapon" prompt
- (2) gets user's input from the kbd
- (3) assigns that input to the variable user

(3) Punctuation used in more than one way: ( ), . , =, :
Tear off that page
Pass them in and to the front...

be sure your name's on one...

Take a picture if you'd like to "keep" it

... then turn back into the packet
Create a short text-adventure in Python...

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

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](http://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!
Surroundings

How many distinct surroundings are there?
Surroundings

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

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>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>

*Note:* 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

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

Notes

Start

Step 1

Step 2

Step 3

Step 4

...?

these two rules are a complete Picobot program

current state | surroundings | direction | new state
---|---|---|---
rule (A) | 0 | Nxxx | S | 0
rule (B) | 0 | xxxx | N | 0
Picobot programming ~ *rules*

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:

<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 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.
Picobot programming ~ \textit{rules}

Picobot checks its rules from the top each time. \textit{When it finds a matching rule}, that rule runs.

\textbf{Notes}

\begin{itemize}
 \item \textbf{current state}
 \item \textbf{surroundings}
 \item \textbf{direction}
 \item \textbf{new state}
\end{itemize}

\begin{tabular}{|c|c|c|c|}
\hline
rule (A) & 0 & Nxxx & S & 0 \\
\hline
rule (B) & 0 & xxxx & N & 0 \\
\hline
\end{tabular}

these two rules are a complete Picobot program

---

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!

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

0  x***  N  0

EWS may be wall or empty space

8 surroundings in one rule

that's it!
The Rule is  One step per rule

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.

*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>

There can only be **ONE** rule per situation! and a "situation" is *state and surroundings*

Notes

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

*These two situations COULD BE the same!*
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

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?

*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

facing to the right
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*

\[
\begin{array}{c} 0 \ \text{xE**} \ \rightarrow \ N \ \ 0 \\
\end{array}
\]

(B) INTERSECTION rule

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

\[
\begin{array}{c} 0 \ \rightarrow \ \\
\end{array}
\]

(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*”

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

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!”

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

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.

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

State 2 means "now facing west"

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

Is it working?
- Onward -

Lab/hw

You are not alone!

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

Happy Picobotting!

And, good luck with the **adventure** of Python!