They call that an alien?

Spock *mind-melds* three-eyed aliens!

Provably.
Pair up with someone nearby – answer these questions together...

Name ______________________

Your favorite __________ is ____________.

Your least favorite ____________ is ____________.

Name ______________________

Your favorite __________ is ____________.

Your least favorite ____________ is ____________.

What is something non-Claremont-menu you have in common?

Then, try these Python q's:

(0) Find the 3 tests and 4 blocks here.

(1) What does this code print?

```python
comp = 'rock'
user = 'rock'
if comp == 'rock':
    if user == 'paper':
        print('I win *_*!')
    elif user == 'scissors':
        print('You win.')
    else:
        print('Tie.')
else:
    print('Tie.')
```

(2) As written, what output does this print?

```python
comp = 'rock'
user = 'rock'
if comp == 'rock':
    if user == 'paper':
        print('I win *_*!')
    elif user == 'scissors':
        print('You win.')
    else:
        print('Tie.')
else:
    print('Tie.')
```

(3) Change these inputs to produce a completely correct RPS output here.

(4) How many of the 9 RPS input cases are fully correctly handled here?

(5) What is the smallest number of blocks and tests you'd need for a full game of RPS?

(Extra) What if it were RPS-5, which includes Lizard and Spock? How about RPS-101?
comp = 'rock'
user = 'rock'

if comp == 'rock':
    if user == 'paper':
        print('I win *_*!*')
    elif user == 'scissors':
        print('You win.')
    else:
        print('Tie.')
        print('Ties go to the runner.')
        print(' - and I am running!')

... what if this else block were indented?
What does this program print?

```python
comp = 'rock'
user = 'rock'

if comp == 'rock':
    print('I win *_*!')

if user == 'paper':
    print('You win. ')

else:
    print('An awful tie')
```

"Quiz" ~ problems 3-5
How many possible “input cases” are there?

For how many is this program correct?

How efficient can we be?

For RPS-3? RPS-5? RPS-101?
comp = 'rock'
user = 'rock'

if comp == 'rock':
    print('I win *_*!')

if user == 'paper':
    print('You win.')

else:
    print('An awful tie')

How many possible “input cases” are there?
For how many is this program correct?

How efficient can we be?
For RPS-3?  RPS-5?  RPS-101?
A correct RPS is possible with only if ... elif ... else!

How many possible “input cases” are there? How efficient can we be? For how many is this program correct? For RPS-3? RPS-5? RPS-101?
Lab 0: Check!
Lab lookback...

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

Finished with lab? OK! No need to stay longer
4-4s is eleven!

print("Eleven is", ((factorial(4)+4)/4)+4)

print("Eleven is", 4/.4+4/4)
4-4s is thirteen!

```python
print("Thirteen is", 44/4+sqrt(4))
```

```
print("Thirteen is", (factorial(4)-(4.4/.4)))
```

Andrew

Oluwatomi
Four fours is ~

sometimes too many...
other times too few...

and never enough!

-- Prof. Su
Email help:  

**Start w/ Piazza...**

*for many questions,* Piazza is a great resource:

this link:

this Q&A page
In-person help: "gruTurinG"

every day there are tutoring hours in the LAC lab

don't sign up  just go!
grutors there to provide support...
Tutoring location @ HMC: **LAC**

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

west-side entrance

Coffee

Class

Lots of grutors - all eager to help!

Come by!

My office hours will be there...

LOTS of tutoring hours, as well...
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 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.”

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?*

CS's 6 big questions are here.

But only one is **programming**. Do you see which?

Can you solve this problem? (CS)

Can you create a process to solve such problems? (programming + CS)

How quickly can you find solutions? (CS)

Do you have the “best” solution? (CS)

Is every problem solvable? (CS)

Is there a way to tell? (CS) 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!
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':  
  print 'Zach Quinto...?\nNo?\nOh.'

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

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       \m
       \7
    print 'I am offline. Try later.'
elif name == 'Zach':
    print 'Zach Quinto...? 
        No?  
    print '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 and have a human-desired output.
The *foreign language* of python...

<table>
<thead>
<tr>
<th>syntax</th>
<th>semantics</th>
<th>intent</th>
</tr>
</thead>
<tbody>
<tr>
<td>How it looks</td>
<td>What it does</td>
<td>What it should do</td>
</tr>
</tbody>
</table>

```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: 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\'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**

How it looks

**semantics**

What it does

**intent**

What it should do
Flowchart...

a graphical view of a program's semantics

prompt and user input

→

name

decision

→

Is name equal to 'Eliot' or 'Ran'

decision

→

Is name equal to 'Zach'

machine-produced output

YES

→

print message for Eliot or Ran

NO

→

print message for everyone else

YES

→

print message for Zach
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**: How 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...?!
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

?
import random  # I can't guess

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!")
(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!')
else:
    print('are these equal?')
```

(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!

```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!')
```

(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: () . ' = , :
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?
**Goal:** full-room coverage with only local sensing...

HW problems 3 and 4: Picobot!

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

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?

$2^4 = 16$ 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>move S</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>xxxx</td>
<td>move 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.
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 ~ 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 ~ *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) 0</td>
<td>Nxxx</td>
<td>S</td>
<td>0</td>
</tr>
<tr>
<td>rule (B) 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>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.

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:

I only care about NORTH being EMPTY

N must be empty

EWS may be wall or empty space

8 surroundings in one rule

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

```plaintext
<table>
<thead>
<tr>
<th>state</th>
<th>surr.</th>
<th>move</th>
<th>new state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N***</td>
<td>-&gt;</td>
<td>W 1</td>
</tr>
<tr>
<td>0</td>
<td>x***</td>
<td>-&gt;</td>
<td>N 0</td>
</tr>
<tr>
<td>1</td>
<td>***x</td>
<td>-&gt;</td>
<td>S 1</td>
</tr>
</tbody>
</table>
```

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>

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?
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?

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*”

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

(B) INTERSECTION rule

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

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

(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*"  

<table>
<thead>
<tr>
<th>State</th>
<th>xE**</th>
<th>-&gt;</th>
<th>N</th>
<th>0</th>
</tr>
</thead>
</table>

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!*"  

<table>
<thead>
<tr>
<th>State</th>
<th>x**</th>
<th>-&gt;</th>
<th>E</th>
<th>1</th>
</tr>
</thead>
</table>

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.  

<table>
<thead>
<tr>
<th>State</th>
<th>NE**</th>
<th>-&gt;</th>
<th>X</th>
<th>2</th>
</tr>
</thead>
</table>

State 2 means "now facing west"  

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

Is it working?
You are not alone!

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

Happy Picobotting!

And, good luck with the adventure of Python!