

# Welcome to CS 115

## Introduction to Computer Science

Handout (read it!):

- Course syllabus



Unofficial alien of our course

# Welcome to CS 115

---

- **Versus CS 110 and CS 181**
  - For 115 you should have some programming experience in some language (e.g. AP comp sci using Java, or CS 110)
  - If you have no programming experience at all, go to CS 110
  - If you have lots of programming experience, go to CS 181 (go to class and ask permission of instructor)
- **Today...**
  - Course overview and logistics
  - What will we do in this course?
  - Getting started

# Acknowledgment to HMC

---

- Course authors

- Eliot Bush (**Green**)



+ Geoff Kuenning

- Zach Dodds (**Gold**)



- Ran (“Ron”) Libeskind-Hadas (**Black** and **Green**)

- Christine Alvarado (UC San Diego)



- Harvey Mudd College CS 5

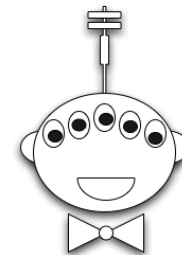
- Three parallel courses, nationally recognized math/sci/engr innovation
  - Interesting, problem-oriented, using Python language
  - Textbook/site under trial at Stevens, Bucknell, elsewhere

- Credit to them -- blame and suggestions to us

# Overview

---

- Weeks 1-5: Thinking functionally/algorithmically
- Weeks 6-8: Computer organization
- Weeks 9-13: Oops! (Object oriented programs)
- Weeks 14-15: some catch-up & projects



15 weeks of  
action-packed  
excitement!

# CS 115 FAQ

---

Q: How much overlap is there between CS 115 and AP CS?

A: Very little (perhaps 10% of the material).

Q: I haven't had AP CS, but I've done some programming. Will I be OK?

A: Quite likely, but feel free to chat with me if you're concerned.

Q: How do I succeed in the course?

A: Make a habit of doing programming exercises every day so you become *fluent*: able to write correct code on paper, without computer.

# Some Logistics...

---

- ☑ All software in this course is freely downloadable
- ☑ Textbook in ebook format
- ☑ Homework assigned/submitted in Canvas
  - Usually due on Wednesday at 11:55 PM
  - Late policy: see syllabus
- ☑ Honor code
  - Individual work unless stated otherwise
- ☑ Closed labs (ask now what that means)



First lab is this Thurs.  
Register for D110B exams ???

# Syllabus/Policies/Advice

---

Bring laptops to class.

If you fail to use laptops properly (i.e. Facebook, messaging, etc.), they will be banned from class.

Your laptop can't write programs.

If you can program, you can write programs on paper too!



What's paper?

# Human Resources

---

- ✓ Instructor: Brian Borowski
- ✓ Teaching Assistants (TAs): Ashley Bromiley, Zach Caldarola, Greg Goldshteyn, Ben Knutson, Ammar Mustafa, Andy Wiggins
- ✓ We'll all have office hours and be available for help
- ✓ Tutors (Academic Support Center)
- ✓ Classmates, friends, enemies ([iDoYourHomework.com](http://iDoYourHomework.com))



Space aliens might help too.



# Programming Languages...

---

---

- \* A+
- \* A++
- \* A#
- \* A-0 programming language
- \* ABAP
- \* ABC
- \* ABC ALGOL
- \* ABLE
- \* ABSET
- \* ABSYS
- \* ACC
- \* Accent
- \* ACT-III
- \* ATOLL - Acceptance, Test Or Launch Language
- \* Action!
- \* ACS
- \* ActionScript
- \* Actor
- \* Ada

## 2000± languages omitted

- \* YAFL
- \* Yellow - Rejected prototype for Ada
- \* Yorick
- \* Y Language
- \* Z notation - A program specification language, like UML.
- \* ZPL
- \* ZZT-oop
- \* ZOPL
- \* ZUG

# Befunge!

(Funge, Argh!, Befreak, Numberix, Weird)

## Code sample: Befunge

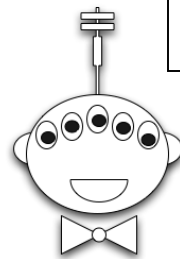


Befunge is a synthetic language which may be useful for something one day, but I never hope to see any such day myself. The befunge interpreter starts reading the program at the upper left corner and moves toward the right, but it may be reversed with a < or sent upwards ^ and down v at will. Arithmetic operations use direct operands and a single stack that reminds me of the unix utility **dc**.

This program prints out the stirring lyrics of the famous camp song "99 Bottles of Beer on the Wall":

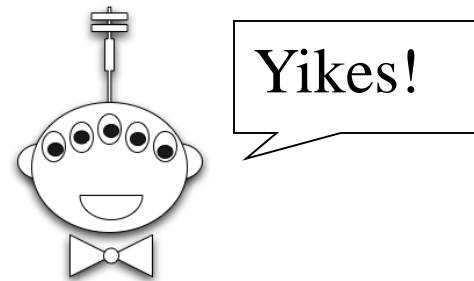
```
9::*\2*+00p0v"."0<
>310p0",">"llaw eht no "v >#v_ ^
^_210p0"--:" v ,
: v " of beer" < :
- >"selttob"00g.^ <
1 >00gl-#^_" elttob erom en0" ^
>00g#^_" selttob erom oN" ^
^_110p0",dnuora ti ssap ,nwod eno ekaT"^
^:-1_010p00gl-00pvv:-lg01_@#g00,*25<
^<
```

This makes  
my eyes  
water!



# Intercal

```
DO ,1 <- #13
PLEASE DO ,1 SUB #1 <- #234
DO ,1 SUB #2 <- #112
DO ,1 SUB #3 <- #112
DO ,1 SUB #4 <- #0
DO ,1 SUB #5 <- #64
DO ,1 SUB #6 <- #194
DO ,1 SUB #7 <- #48
PLEASE DO ,1 SUB #8 <- #22
DO ,1 SUB #9 <- #248
DO ,1 SUB #10 <- #168
DO ,1 SUB #11 <- #24
DO ,1 SUB #12 <- #16
DO ,1 SUB #13 <- #214
PLEASE READ OUT ,1
PLEASE GIVE UP
```



## 1.1 ORIGIN AND PURPOSE

The INTERCAL programming language was designed the morning of May 26, 1972 by Donald R. Woods and James M. Lyon, at Princeton University. Exactly when in the morning will become apparent in the course of this manual. It was inspired by one ambition; to have a compiler language which has nothing at all in common with any other major language. By 'major' was meant anything with which the authors were at all familiar, e.g., FORTRAN, BASIC, COBOL, ALGOL, SNOBOL, SPITBOL, FOCAL, SOLVE, TEACH, APL, LISP, and PL/I. For the most part, INTERCAL has remained true to this goal, sharing only the basic elements such as variables, arrays, and the ability to do I/O, and eschewing all conventional operations other than the assignment statement (FORTRAN "=").

## 1.2 ACRONYM

The full name of the compiler is "Compiler Language With No Pronounceable Acronym", which is, for obvious reasons, abbreviated "INTERCAL".

## 1.3 ACKNOWLEDGMENTS

# Python

- Relatively “nice” syntax
- Emerging as language of choice in many fields
- Packages for graphics, audio, scientific computing, ...

## Python

```
print "Hello World"
```

## Java

```
class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```

## Befunge

```
>                                v  
v  ,,,,,,"Hello"<  
>48*,                                v  
v,,,,,,,"World!"<  
>25*,@
```

Prof. Borowski takes on Python



# Hello World...

```
#include <iostream>
```

```
using namespace std;
```

C++

```
int main()
```

```
{
```

```
    cout << "Hello World!" << endl;
```

```
    return 0;
```

```
}
```

Ook. Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.  
Ook. Ook. Ook. Ook. Ook! Ook? Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.  
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook? Ook! Ook! Ook? Ook! Ook? Ook.  
Ook! Ook. Ook. Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.  
Ook. Ook. Ook! Ook? Ook? Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.  
Ook! Ook! Ook? Ook! Ook? Ook. Ook. Ook. Ook! Ook. Ook. Ook. Ook. Ook. Ook. Ook.  
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook! Ook. Ook! Ook. Ook. Ook. Ook.  
Ook. Ook. Ook! Ook. Ook. Ook? Ook. Ook? Ook. Ook? Ook. Ook. Ook. Ook. Ook. Ook.  
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook! Ook? Ook? Ook. Ook. Ook.  
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.  
Ook. Ook? Ook! Ook! Ook? Ook! Ook? Ook. Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook.  
Ook? Ook. Ook? Ook. Ook? Ook. Ook? Ook. Ook! Ook. Ook. Ook. Ook. Ook. Ook. Ook.  
Ook! Ook. Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook.  
Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook! Ook!  
Ook! Ook. Ook. Ook? Ook. Ook? Ook. Ook. Ook. Ook! Ook. Ook! Ook? Ook! Ook! Ook?  
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.  
Ook. Ook. Ook. Ook. Ook! Ook.

Ook

# Some things you'll do this semester...

## Sequence alignment

ATTATCG

ACATTC

A~~T~~TATCG → Delete T

A ~~T~~ATCG → Change T to C

A CAT\_ CG → Insert T here

A CATTC~~G~~ → Delete G

A CATTC

Distance is 4

A~~TT~~AT-~~C~~G

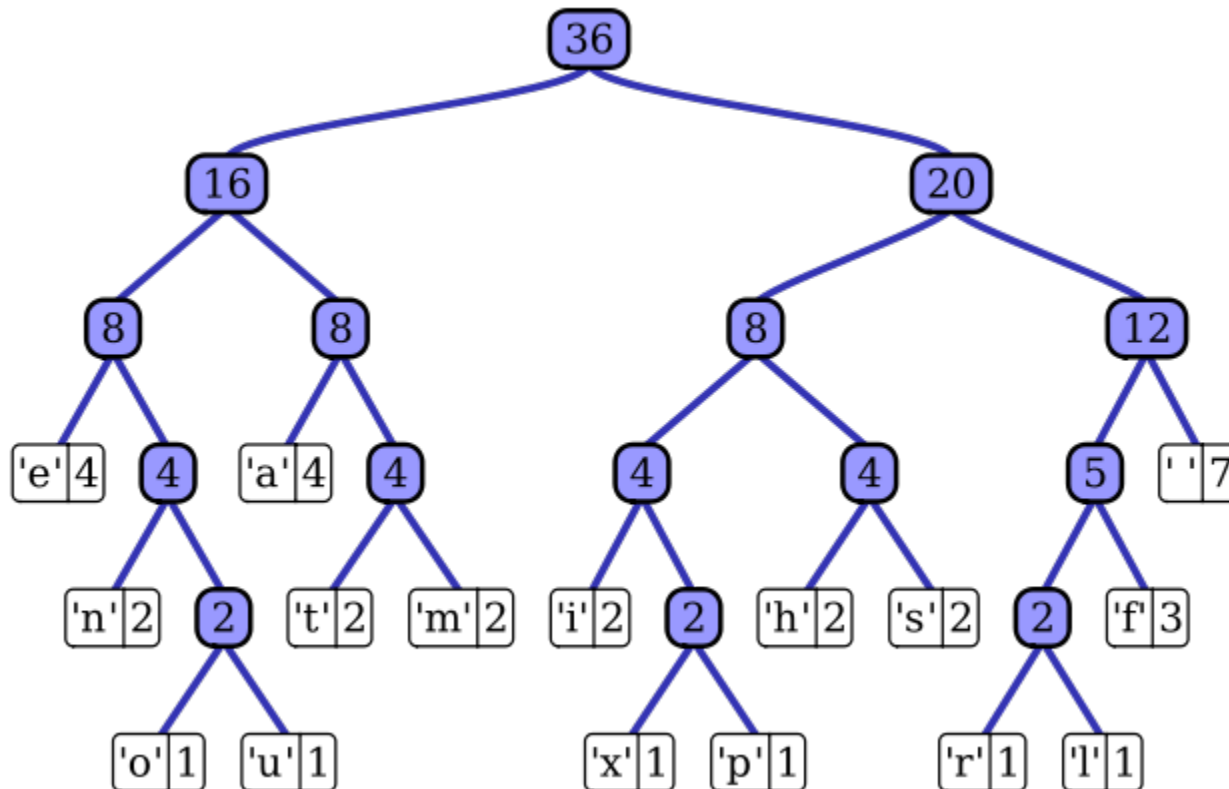
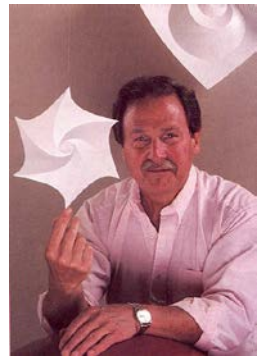
A-~~C~~AT~~T~~C-

# Spel Cheking...

© Original Artist  
Reproduction rights obtainable from  
[www.CartoonStock.com](http://www.CartoonStock.com)



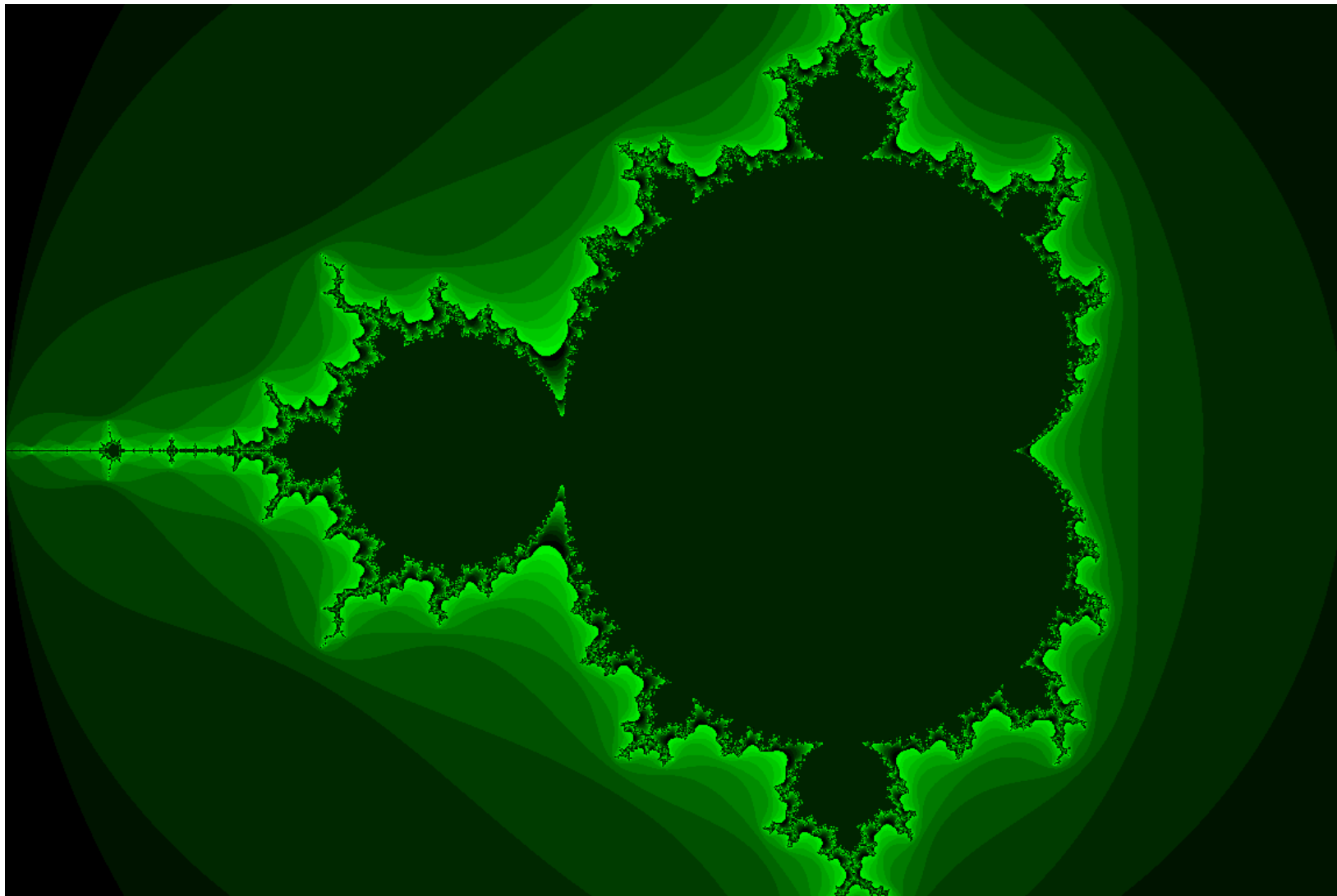
# Huffman Data Compression



Char	Freq	Code
space	7	111
a	4	010
e	4	000
f	3	1101
h	2	1010
i	2	1000
m	2	0111
n	2	0010
s	2	1011
t	2	0110
l	1	11001
o	1	00110
p	1	10011
r	1	11000
u	1	00111
x	1	10010



# Mandelbrot Set

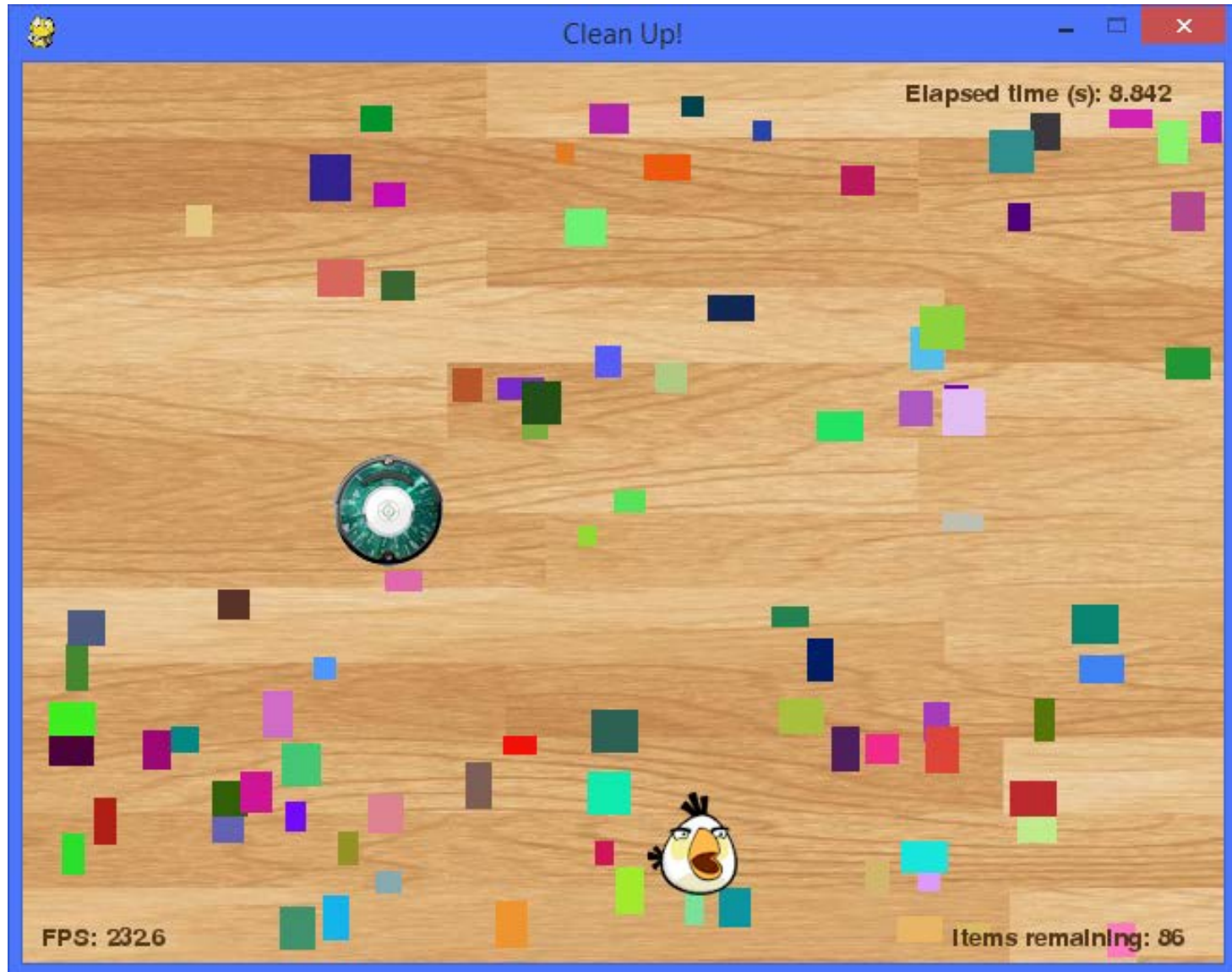


# Connect 4 AI - maybe



Alien Intelligence?

# Clean Up! - maybe



# Picobot!

<http://www.cs.hmc.edu/picobot/>

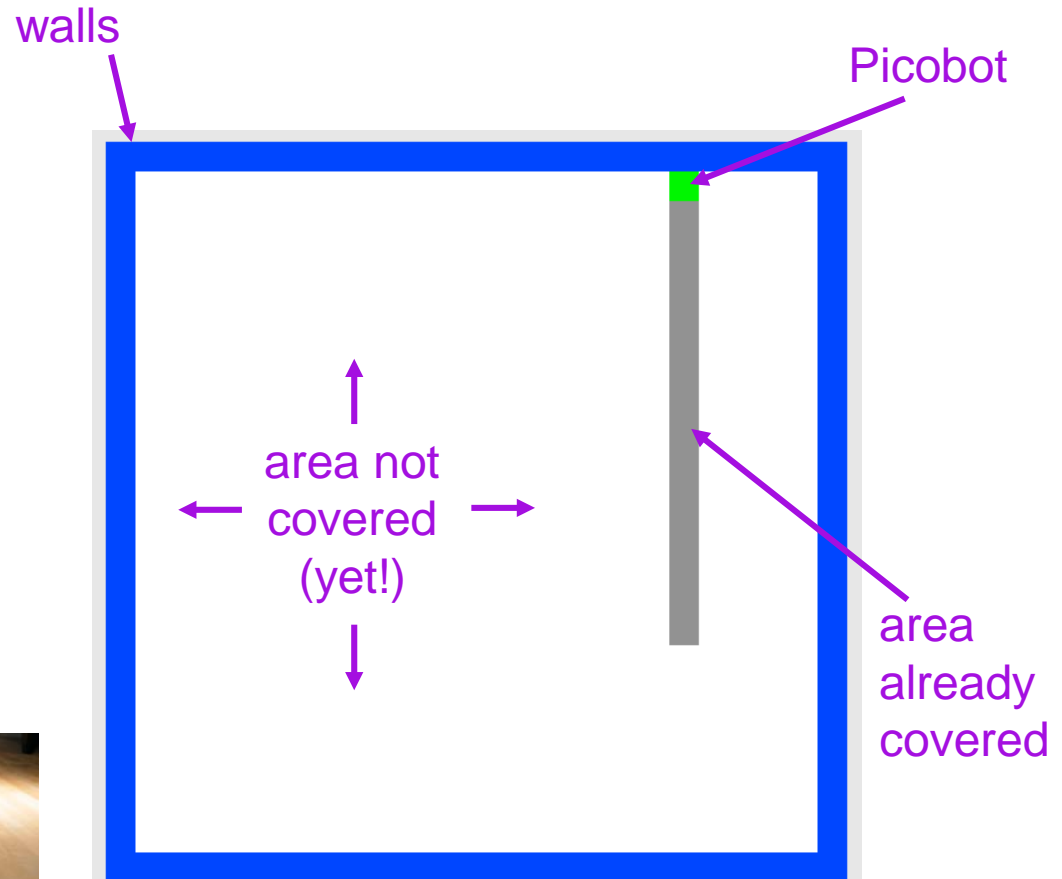
Reading: Chapter 1 in the book



Murata Girl



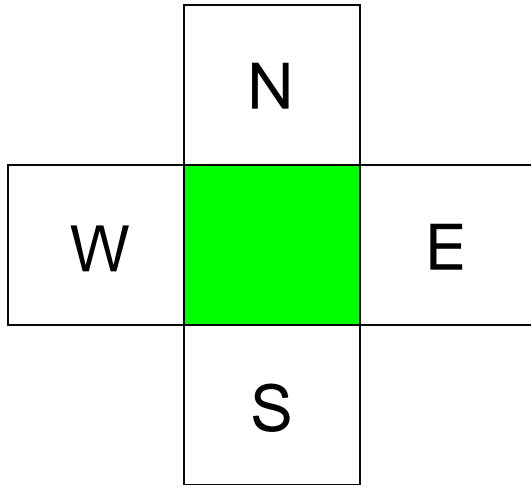
Roomba



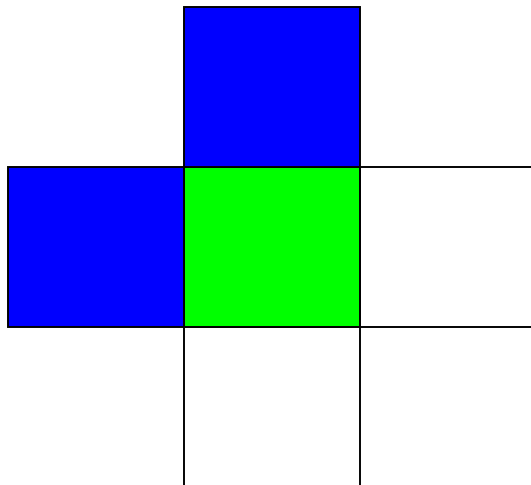
**Goal:** whole-environment coverage  
with only *local sensing*...

DEMO!

# Environment in the NEWS!



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



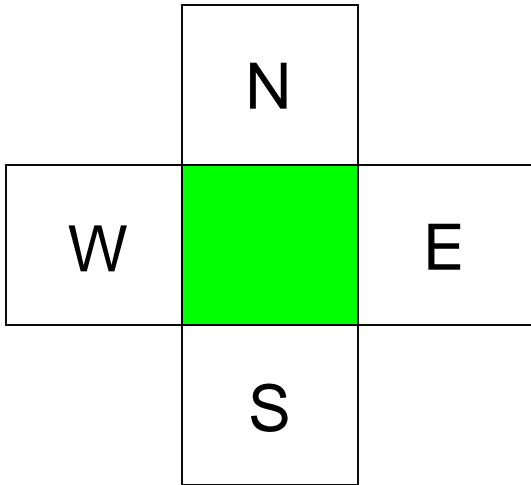
For example, here its surroundings are

**NxWx**

↑   ↑   ↑   ↑  
N   E   W   S

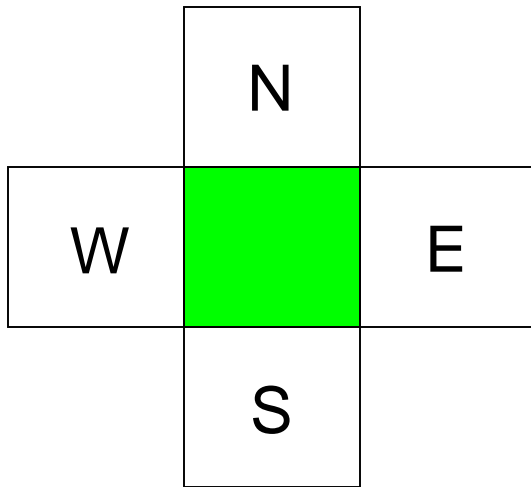
Surroundings are  
always in NEWS order.

# Surroundings



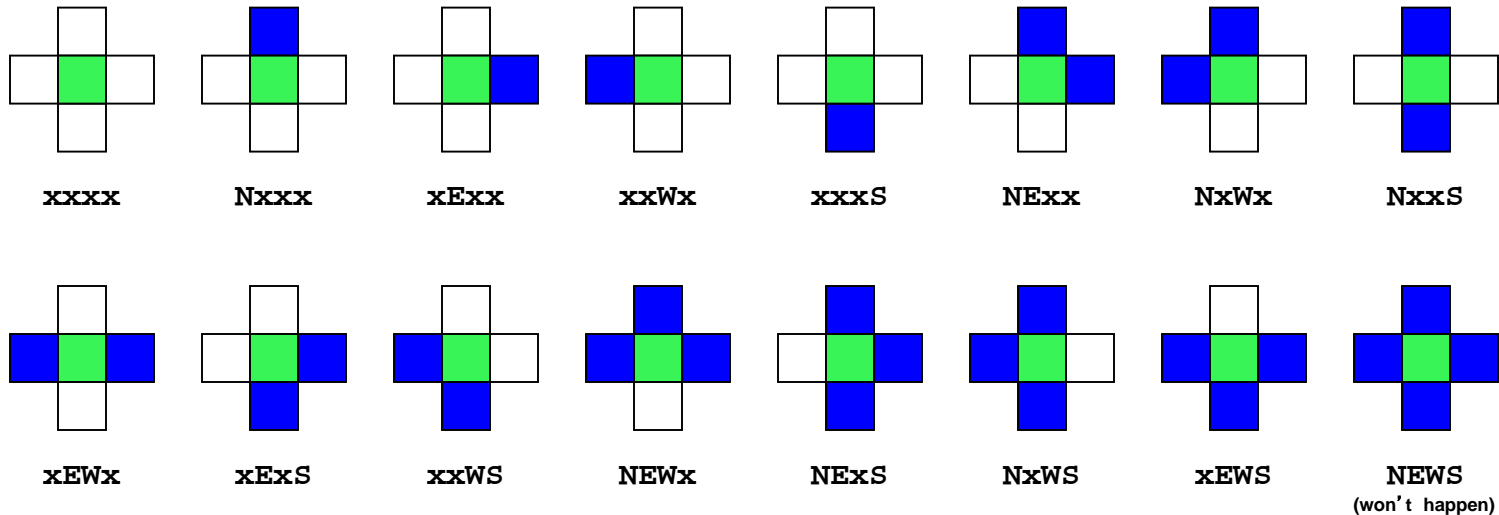
How many distinct  
surroundings are there?

# Surroundings

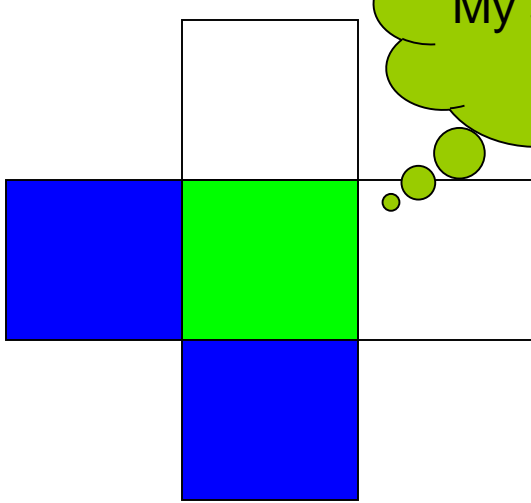


How many distinct  
surroundings are there?

$2^4 == 16$  possible ...



# State



I am in state 0.  
My surroundings  
are xxWS.

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

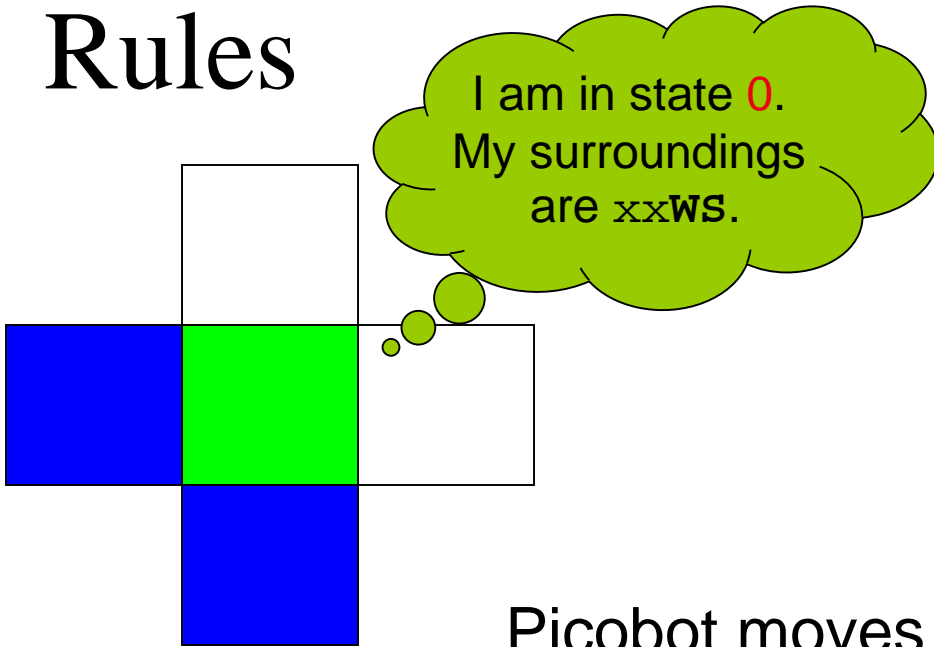
**State** is the *internal context* of computation.

Picobot always starts in **state 0**.

**State** and **surroundings** represent everything the robot knows about the world



# Rules



Aha!

I should move N.

I should enter state 0.

Picobot moves according to a set of rules:

state

surroundings

0

xxWS



direction

new state

N

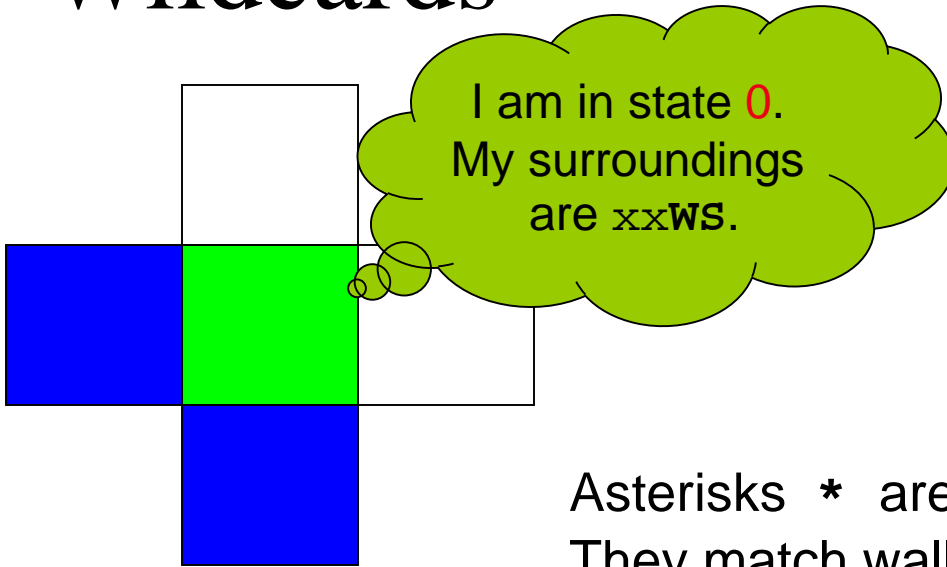
0

A capital "X"  
here means  
"Don't Move"

*If I'm in state 0  
seeing xxWS,*

*Then I move **N**orth, and  
change to state 0.*

# Wildcards



*Aha! This matches  $x***$*

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

state	surroundings		direction	new state
-------	--------------	--	-----------	-----------

0

$x***$



N

0

*and EWS may be wall or empty space*

*N must be empty*

# What will this set of rules do to Picobot?

A capital "X"  
here means  
"Don't Move"

state	surroundings		direction	new state
0	<b>x***</b>	->	N	0
0	<b>N***</b>	->	X	0

Add some code here to make Picobot go up and down in the same column forever!

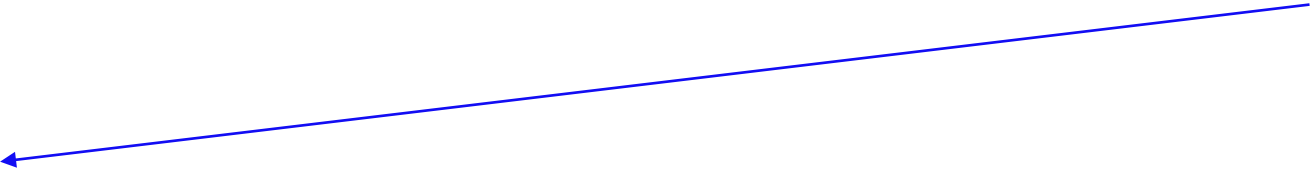
Picobot checks its rules from the top each time.

When it finds a matching rule, that rule runs.

Only one rule is allowed per state and surroundings.

What will this set of rules do to Picobot?

state	surroundings		direction	new state
0	<b>x***</b>	->	N	0
0	<b>N***</b>	->	X	<b>1</b>
1	<b>***x</b>	->	S	1
1	<b>***S</b>	->	X	0



Picobot checks its rules to find one that applies

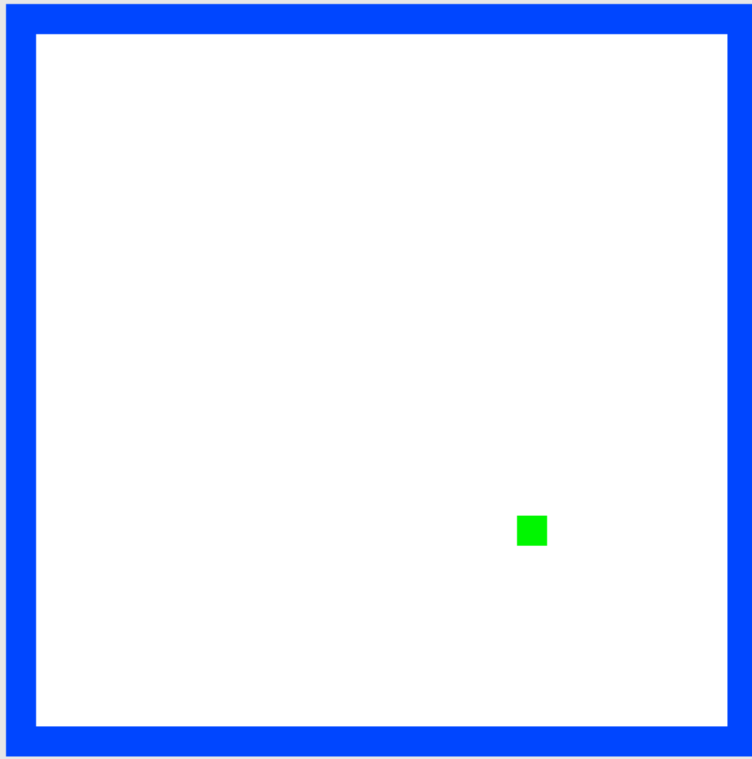
When it finds a matching rule, that rule runs.

Only one rule is allowed per state and surroundings.

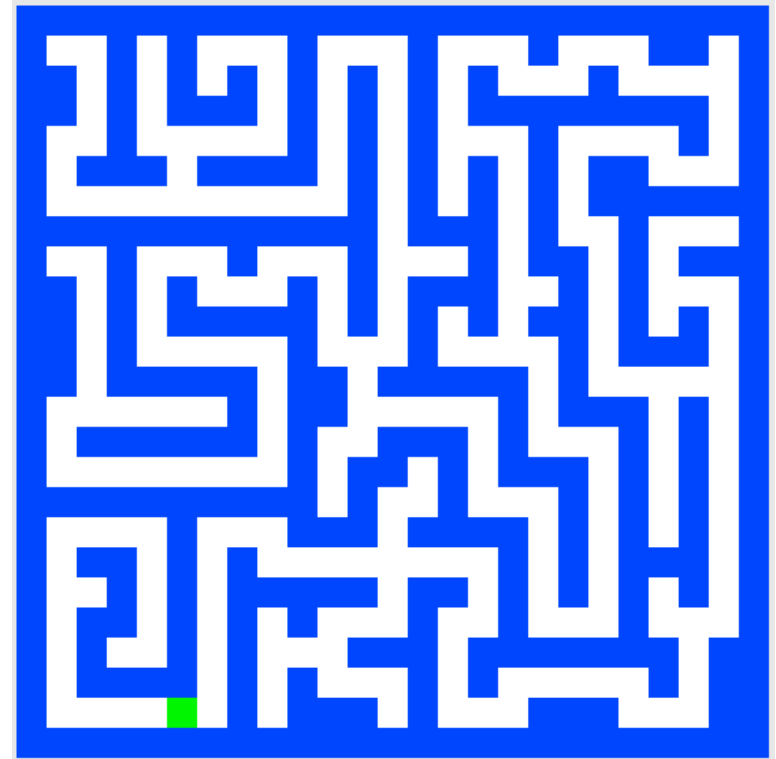
# This week!

Write rules that will always cover these two rooms.  
(*separate* sets of rules are encouraged...)

## Lab Problem



## Problem 2



Your “program” can be slow but it should work for any starting location and for any wall-connected maze!

**DEMO!**

**our best:** 3 states, 7 rules

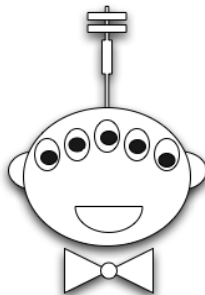
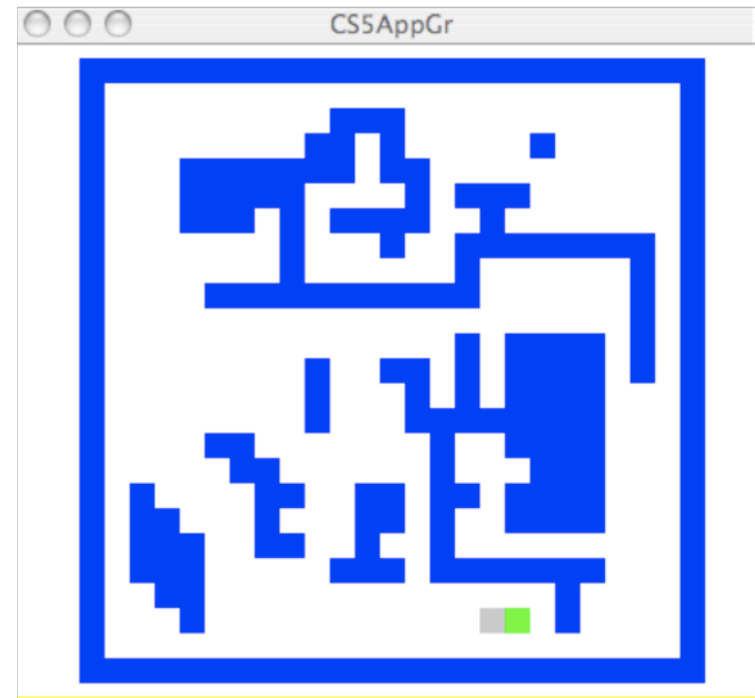
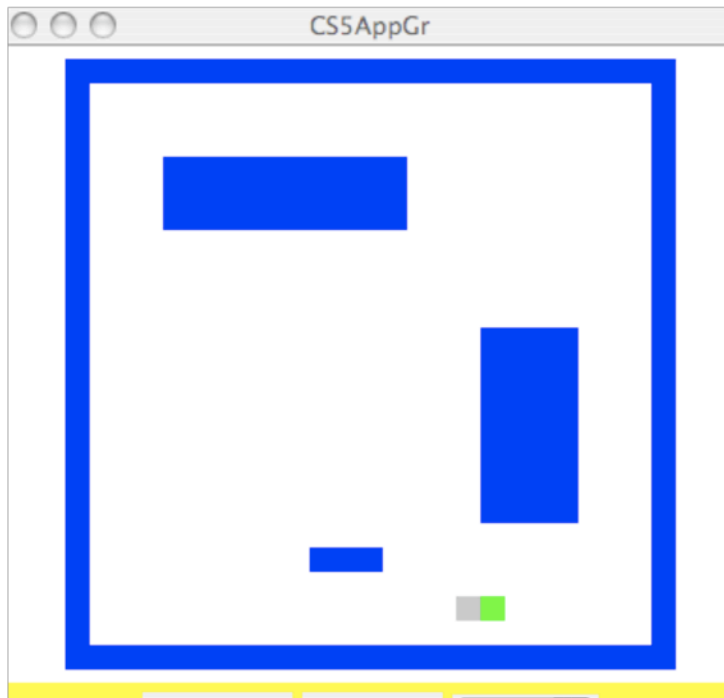
**our best:** 4 states, 8 rules

# What's the point?

---

- Simple syntax can support “powerful” computation:  
The picobot language syntax is very simple yet it can control a robot in a complex environment.
- Computer scientists examine limitations of languages:
  - Are there environments that the picobot language cannot navigate?
  - If so, what features could be added to give the language more “power”?

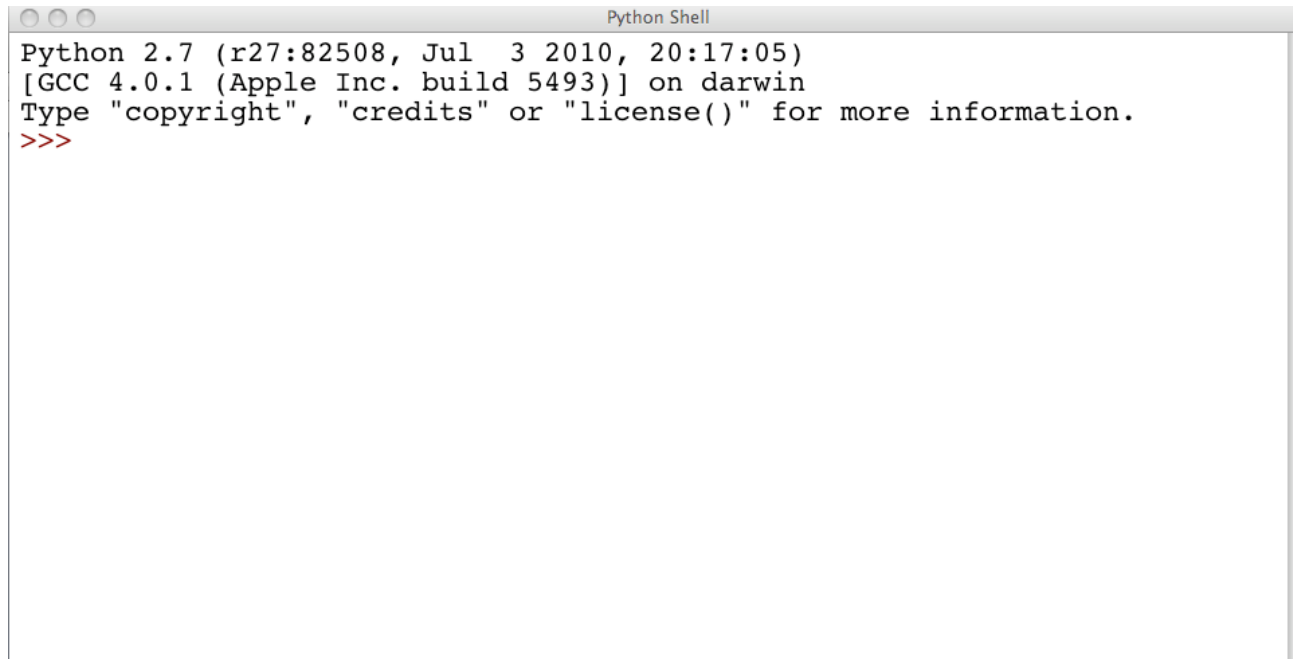
# How about rooms in general?



Picobot has 100 states,  
but the “room” could be  
arbitrarily big and weird!

# Python and IDLE

---



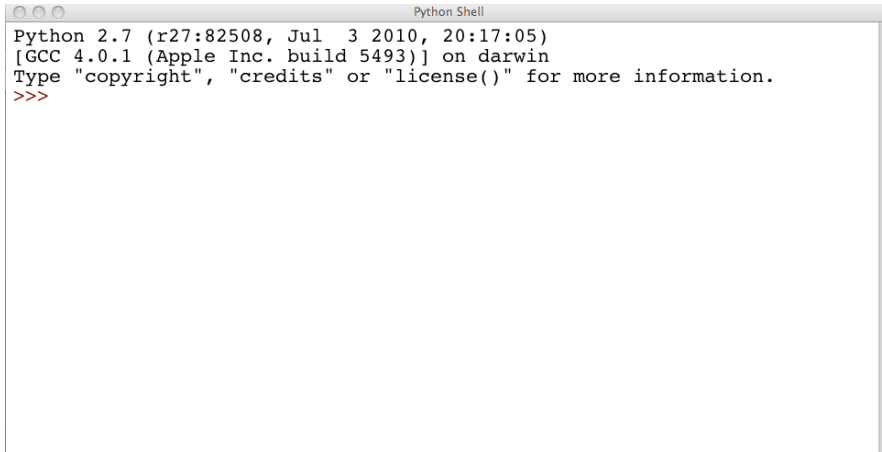
```
Python Shell
Python 2.7 (r27:82508, Jul  3 2010, 20:17:05)
[GCC 4.0.1 (Apple Inc. build 5493)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
```

Shell window: Interacting with  
Python!



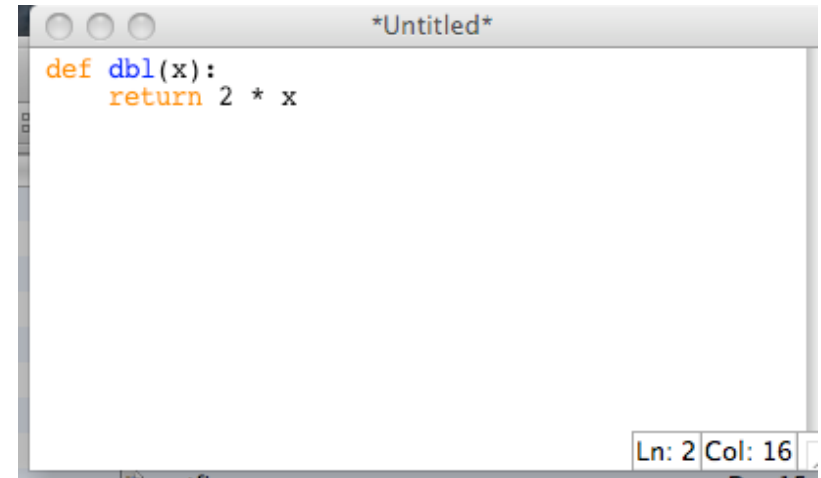
# Python and IDLE

---

A screenshot of a Python Shell window titled "Python Shell". The window shows the Python 2.7 version information and the GCC compiler details. The prompt is >>>.

```
Python 2.7 (r27:82508, Jul 3 2010, 20:17:05)
[GCC 4.0.1 (Apple Inc. build 5493)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
```

Shell window: Interacting with Python

A screenshot of an IDLE Editor window titled "\*Untitled\*". The window shows a Python function definition. The prompt is >>>.

```
def dbl(x):
    return 2 * x
```

Editor: Writing your own functions!

Invoke through “File” and then “New Window”

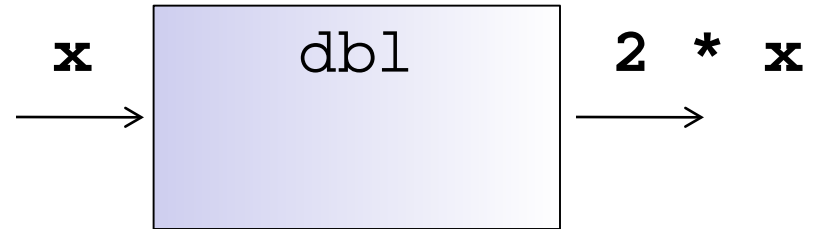
Save

Hit F5 to run

# Defining your own functions!

---

```
def dbl(x):  
    return 2 * x
```



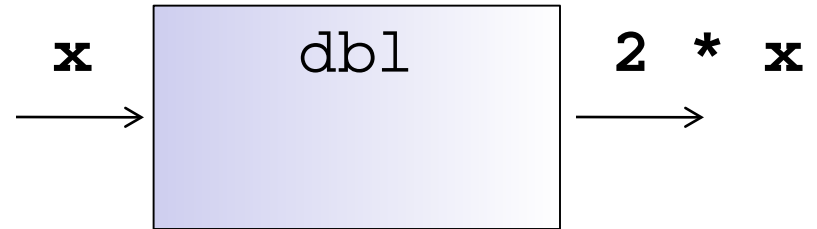
Notice the indentation. This is done using “tab” and it’s absolutely necessary!



# Defining your own functions!

---

```
def dbl(x):  
    return 2 * x
```



```
def dbl(myInput):  
    myOutput = 2 * myInput  
    return myOutput
```

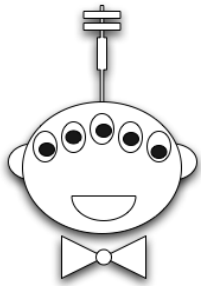
Notice the indentation. This is done using “tab” and it’s absolutely necessary!



# Docstrings!

---

```
def dbl(x):  
    """This function takes a number x as input  
    and returns 2 * x"""  
    return 2 * x
```



This is sort of like teaching  
your programs to talk to  
you!

# Docstrings... and comments

---

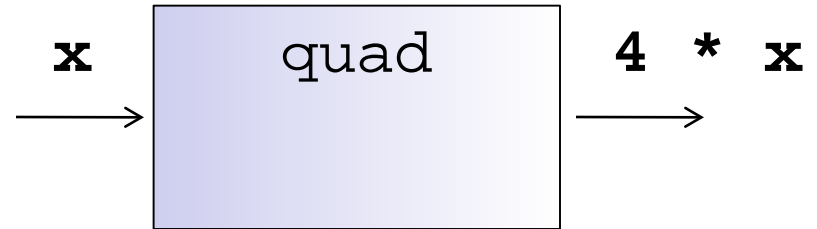
```
# Doubling program
# Authors: Ran Libeskind-Hadas
# Date:      August 27, 2011

def dbl(x):
    """This function takes a number x as input
    and returns 2 * x"""
    return 2 * x
```

# Composition of functions

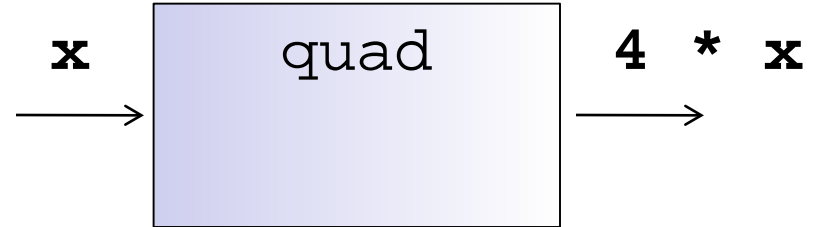
---

```
def quad(x):  
    return 4 * x
```

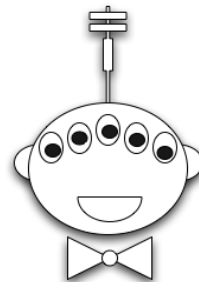


# Composition of functions

```
def quad(x):  
    return 4 * x
```



```
def quad(x):  
    return dbl(dbl(x))
```

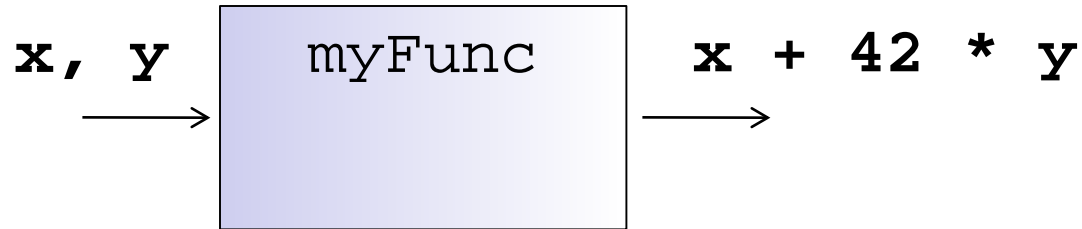


Doubly cool! (draw the boxes)



# Multiple inputs...

---



```
# myFunc
# Authors: Ran Libeskind-Hadas
# Date:    August 27, 2012
```

```
def myFunc(x, y):
    """returns x + 42 * y"""
    return x + 42 * y
```



That's a kind  
of a funky  
function!



# Mapping with Python...

---

```
def dbl(x):  
    """ returns 2 * x """  
    return 2 * x
```

```
>>> map(dbl, [0, 1, 2, 3, 4])  
[0, 2, 4, 6, 8]
```

```
def evens(n):  
    myList = range(n)  
    doubled = map(dbl, myList)  
    return doubled
```

Alternatively....

```
def evens(n):  
    return map(dbl, range(n))
```

# reduce-ing with Python...

---

```
def add(x, y):  
    """returns x + y"""  
    return x + y
```

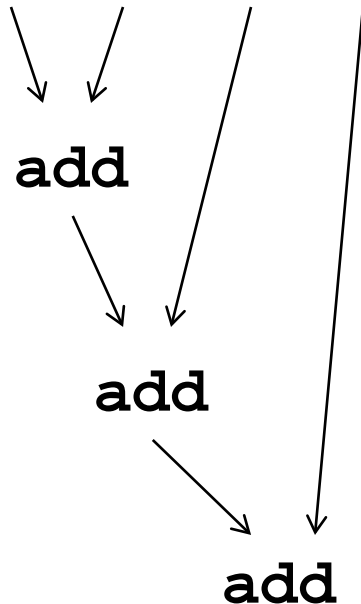
```
>>> reduce(add, [1, 2, 3, 4])  
10
```

# reduce-ing with Python...

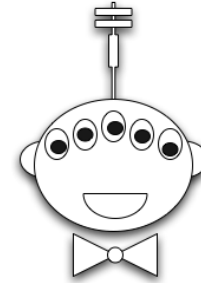
---

```
def add(x, y):  
    """returns x + y"""  
    return x + y
```

```
>>> reduce(add, [1, 2, 3, 4])
```



# Google's "Secret"



This is what  
put Google on  
the map!



Research Publications

## **MapReduce: Simplified Data Processing on Large Clusters**

[Jeffrey Dean](#) and [Sanjay Ghemawat](#)

### **Abstract**

MapReduce is a programming model and an associated implementation for processing intermediate key/value pairs, and a reduce function that merges all intermediate values paper.

Programs written in this functional style are automatically parallelized and executed on scheduling the program's execution across a set of machines, handling machine failure

# Try this...

---

Write a function called `span` that returns the difference between the maximum and minimum numbers in a list...

```
>>> span([3, 1, 42, 7])
```

```
41
```

```
>>> span([42, 42, 42, 42])
```

```
0
```

```
min(x, y)
```

```
max(x, y)
```

These are built-in to Python!

# Try this...

---

1. Write a Python function called `gauss` that takes as input a positive integer  $N$  and returns the sum  
 $1 + 2 + \dots + N$



2. Write a python function called `sumOfSquares` that takes as input a positive integer  $N$  and returns the sum  
 $1^2 + 2^2 + 3^2 + \dots + N^2$



You can write extra  
“helper” functions too!