Welcome to CS 5 Black!

Professor: Ran Libeskind-Hadas aka “Prof. Ran” (“RON”)
E-mail: ran@cs.hmc.edu
Office: Olin 1253

Three handouts today:
Lecture notes
Syllabus
Worksheet

Some reviews of CS 5 Black…
- “Of all the courses that we’ve seen, this was definitely one of them!” – Rotten Tomatoes
- “We don’t have strong enough words to describe this course!” – NY Times

Official alien of CS 5 Black
Worksheet

• Your name (that you prefer to use in class)
• What do you like to do for fun?
• Something “random” about you!

Please give us a “snappy”!
Worksheet

• Your name (that you prefer to use in class)
  – Ran “RON” Libeskind-Hadas

• What do you like to do for fun?
  – Mountain biking, hiking, cooking

• Something “random” about you!
  – Hmmm…
What is this course about?

- **Big and beautiful ideas in Computer Science**
  - Recursion (self-referential code!)
  - Memoization (making recursion fast!)
  - How computers work!
  - Classic algorithms (e.g., shortest paths)
  - Object-oriented programming
  - AI techniques
  - What are the fundamental limits of what computers can do?
Some things *you’ll* do this semester…

- Spell checker *(demo)*
- Data compression
- Secret sharing
- Connect 4 AI *(demo)*
- … lots more stuff …
- Final project

14 weeks of action-packed adventure!
Please read the syllabus

Silly bus?
A few highlights from the

• Come to class, come to labs
  – Class attendance is required; if you’re going to miss class, let me know in advance
  – Labs are optional, but incentivized and encouraged!

• Please no use of electronic devices in class, but laptops welcome in lab!

• Grading:
  – In-class worksheets
  – Weekly assignments, due Monday nights at 11:59 PM
  – Midterm and final

• Late homework policy (3 Euros in the CS 5 Bank!)

• Accommodations :^)

if Mudder:
P/HP/NC
derse:
letter grade

Is the bank FDIC insured?
CS 5 Gold and Black, One-Stop-Shopping!

www.cs.hmc.edu/cs5
Online (interactive) “textbook”

CS 5: Welcome!

Homework Assignments and Labs

Lecture Slides

Acknowledgments and thanks...

CS for All

Christine Alvarado (UC San Diego), Zachary Dodds (Harvey Mudd), Geoff Kuenning (Harvey Mudd), Ran Libeskind-Hadas (Harvey Mudd)

To The Reader

Welcome! This book (and course) takes a unique approach to “Intro CS.” In a nutshell, our objective is to provide an introduction to *computer science* as an intellectually rich and vibrant field rather than focusing exclusively on *computer programming*. While programming is certainly an important and pervasive element of our approach, we emphasize concepts
“Individual” versus “Individual or Pair”

Pairs

one computer

tradeoff typing/debugging ~ about every 20 minutes

Partners

two computers

both partners type/debug ~ provide help as needed

Standard is the same either way:

After finishing the hw, (a) each person has contributed equally and (b) both could complete the problems on their own

Submit with a partner as full co-owners of the work.
Honor Code

• You're *encouraged* to **discuss** problems with other students – or tutors - or any instructors.

• You may **not** share written, electronic or verbal solutions with other students, present or past:

  Please *do* use the internet for Python language references
  Please *do* use other's eyes for finding syntax errors.
  Do *not* use the internet (or intranet) to (try to) find solutions...
  If you work as a pair/partners, the rules apply for the duo.

**Sign & submit** CS's honesty policy **online** in this week's lab.
A note about notes…

• The lecture notes in class will have space for you to write stuff
• The full note sets (including solutions to in-class exercises) will be posted on the course website a few hours after class
Picobot!

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

**area already covered**

**area not covered (yet!)**

walls

Murata Girl

Roomba

Reading: Chapter 1 in the book (http://www.cs.hmc.edu/csforall/)
Environment in the NEWS!

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.
How many distinct surroundings are there?

\[2^4 = 16\] possible …
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.
Picobot moves according to a set of rules:

<table>
<thead>
<tr>
<th>state</th>
<th>surroundings</th>
<th>direction</th>
<th>new state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>xxWS</td>
<td>N</td>
<td>0</td>
</tr>
</tbody>
</table>

If I'm in state 0 seeing xxWS,

Then I move North, and change to state 0.

I am in state 0. My surroundings are xxWS.

Aha! I should move N. I should enter state 0.

A capital “X” here means “Don’t Move”
Wildcards

Asterisks * are wildcards. They match walls or empty space:

I am in state 0. My surroundings are xxWS.

Aha! This matches **x***

and EWS may be wall or empty space

N must be empty

<table>
<thead>
<tr>
<th>state</th>
<th>surroundings</th>
<th>direction</th>
<th>new state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td><strong>x</strong>*</td>
<td>→</td>
<td>N</td>
</tr>
</tbody>
</table>
What Will This Set of Rules Do to Picobot?

<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>-&gt;</td>
<td>N</td>
</tr>
<tr>
<td>0</td>
<td>N***</td>
<td>-&gt;</td>
<td>X</td>
</tr>
</tbody>
</table>

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.

A capital “X” here means “Don’t Move”
What Will This Set of Rules Do to Picobot?

<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>-&gt;</td>
<td>N</td>
</tr>
<tr>
<td>0</td>
<td>N***</td>
<td>-&gt;</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>***x</td>
<td>-&gt;</td>
<td>S</td>
</tr>
<tr>
<td>1</td>
<td>***S</td>
<td>-&gt;</td>
<td>X</td>
</tr>
</tbody>
</table>

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!

**our best:** 3 states, 7 rules (but Cam Zhou had 6)  
**our best:** 4 states, 8 rules
How about obstacles?

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

Want to try the "pebble" version? Talk to Ran!
Segue way to Python...

def dbl(x):
    return 2 * x

def dbl(myArgument):
    myResult = 2 * myArgument
    return myResult

Notice the indentation. This is done using “tab” and Python requires it.
def dbl(x):
    """This function takes a number x and returns 2 * x """
    return 2 * x

Notice the three leading and ending double quotes (single quotes are OK too, just be sure to be consistent!)
Docstrings...and comments

# Doubling program
# Author:  Ran Libeskind-Hadas
# Date:  August 27, 2017
# Time Spent: 14 hours

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

>>> help(dbl)
Composition of functions

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

`quad(x)`

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

Doubly cool!
# myFunc
# Author: Ran Libeskind-Hadas
# Date: August 27, 2017

def myFunc(x, y):
    """ returns x + 42 * y """
    return x + 42 * y
def dbl(x):
    """returns 2 * x """
    return 2 * x

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

>>> map(dbl, [0, 1, 2, 3, 4])
<map object at 0x101a9b2b0>

This output is called an “iterator”
Mapping with Python...

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

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

>>> map(dbl, [0, 1, 2, 3, 4])
<map object at 0x101a9b2b0>

>>> range(5)
[0, 1, 2, 3, 4]

>>> range(1, 5)
[1, 2, 3, 4]
Mapping with Python...

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

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

>>> evens(5)
[0, 2, 4, 6, 8]

def evens(n):
    """returns list of first n evens """

Try writing the evens function using map, range, and dbl.
def dbl(x):
    """returns 2 * x"""
    return 2 * x

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

def evens(n):
    """returns list of first n evens """
    myList = range(n)
    doubled = list(map(dbl, myList))
    return doubled

Alternatively….

def evens(n):
    return list(map(dbl, range(n)))
reduce-ing with Python...

```python
from functools import reduce

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

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

reduce-ing with Python...
Google’s “Secret”

This is what put Google on the map!

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 in the 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 in your worksheet…

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

```python
>>> span([3, 1, 42, 7])
41
>>> span([42, 42, 42, 42])
0
```

These are built in to Python!

```python
min(x, y)
max(x, y)
```
Try this in your worksheet…

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

```python
>>> span([3, 1, 42, 7])
41
>>> span([42, 42, 42, 42])
0
```

```python
def span1(myList):
    """ returns difference between largest and smallest element in list """
    smallest = reduce(min, myList)
    largest = reduce(max, myList)
    output = largest - smallest
    return output

def span2(myList):
    """ returns difference between largest and smallest element in list """
    return reduce(max, myList) - reduce(min, myList)
```
Try this in your notes…

1. Write a python function called `gauss(n)` that accepts a positive integer n and returns the sum
   \[1 + 2 + \ldots + n\]

2. Write a python function called `sumSquares(n)` that accepts a positive integer n and returns the sum
   \[1^2 + 2^2 + 3^2 + \ldots + n^2\]

You can use `add` and write other “helper” functions!

```python
def add(x, y):
    """ returns x + y ""
    return x + y
```
def gauss(n):
    """ return sum of numbers 1 through n """
    return reduce(add, range(n+1))

def sumSquares(n):
    return reduce(add, map(square, range(n+1)))

def square(x):
    return x * x

def add(x, y):
    return x+y