CS 106B

Lecture 29: Conclusion

Thursday, August 17, 2017

Programming Abstractions
Summer 2017
Stanford University
Computer Science Department

Lecturer: Chris Gregg

Today's Topics

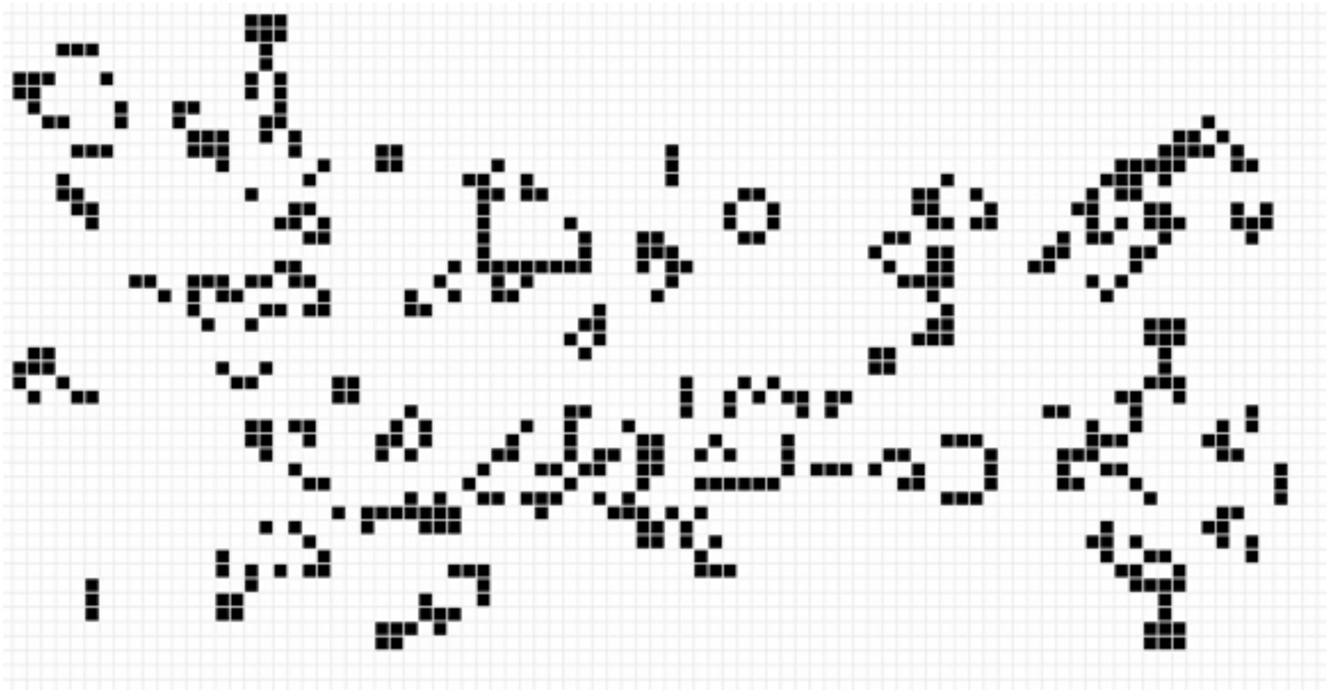
- Logistics
  - Final Exam Saturday, 8:30am. The room is currently scheduled for Dinkelspiel Auditorium, but I have to make sure it will work for laptops that need power
  - Any last minute concerns: please email Chris or Jason

- Where we have been
- Where you are going
  - How to Prepare for CS107

Where We Have Been

CS 106B

Where We Have Been: Game of Life

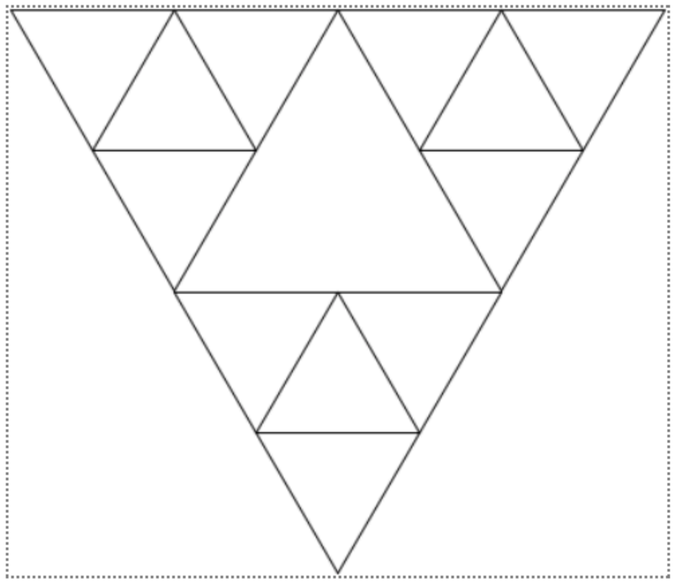


Console

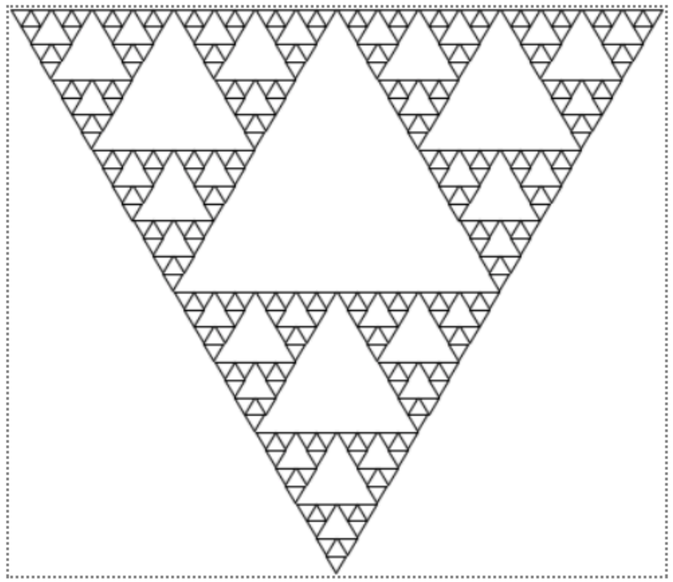
```
Welcome to the CS 106B Game of Life,
a simulation of the lifecycle of a bacteria colony.
Cells (X) live and die by the following rules:
- A cell with 1 or fewer neighbors dies.
- Locations with 2 neighbors remain stable.
- Locations with 3 neighbors will create life.
- A cell with 4 or more neighbors dies.

Grid input file name? simple.txt
Should the simulation wrap around the grid (y/n)? n
---------
---------
---XXX---
---------
---------
a)nimate, t)ick, q)uit? t
---------
----X----
----X----
----X----
---------
a)nimate, t)ick, q)uit? t
---------
---------
---XXX---
---------
---------
a)nimate, t)ick, q)uit? q
Have a nice Life!
```

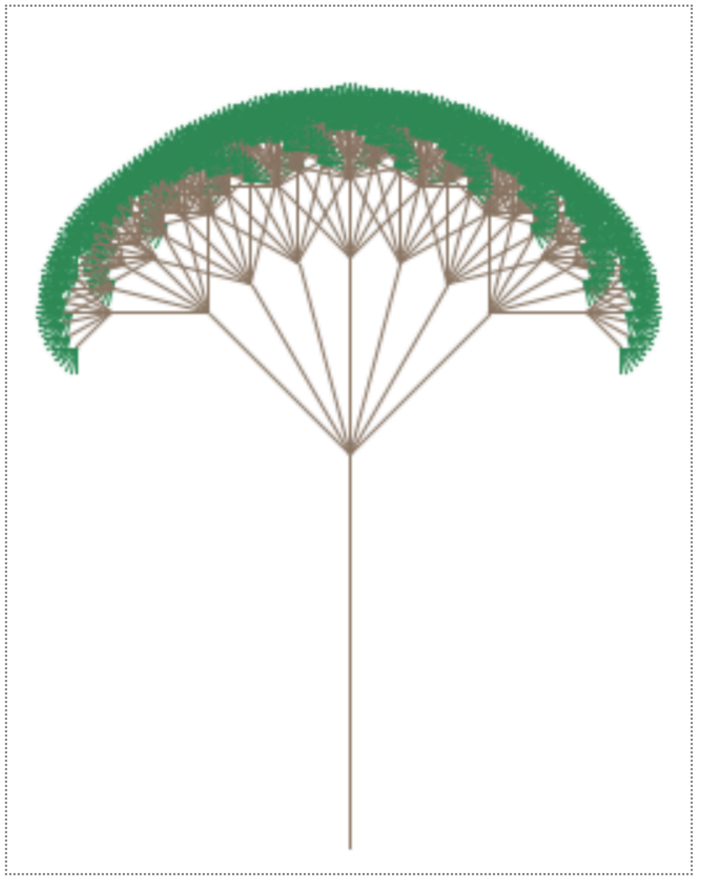
Where We Have Been: Fractals



Order-3

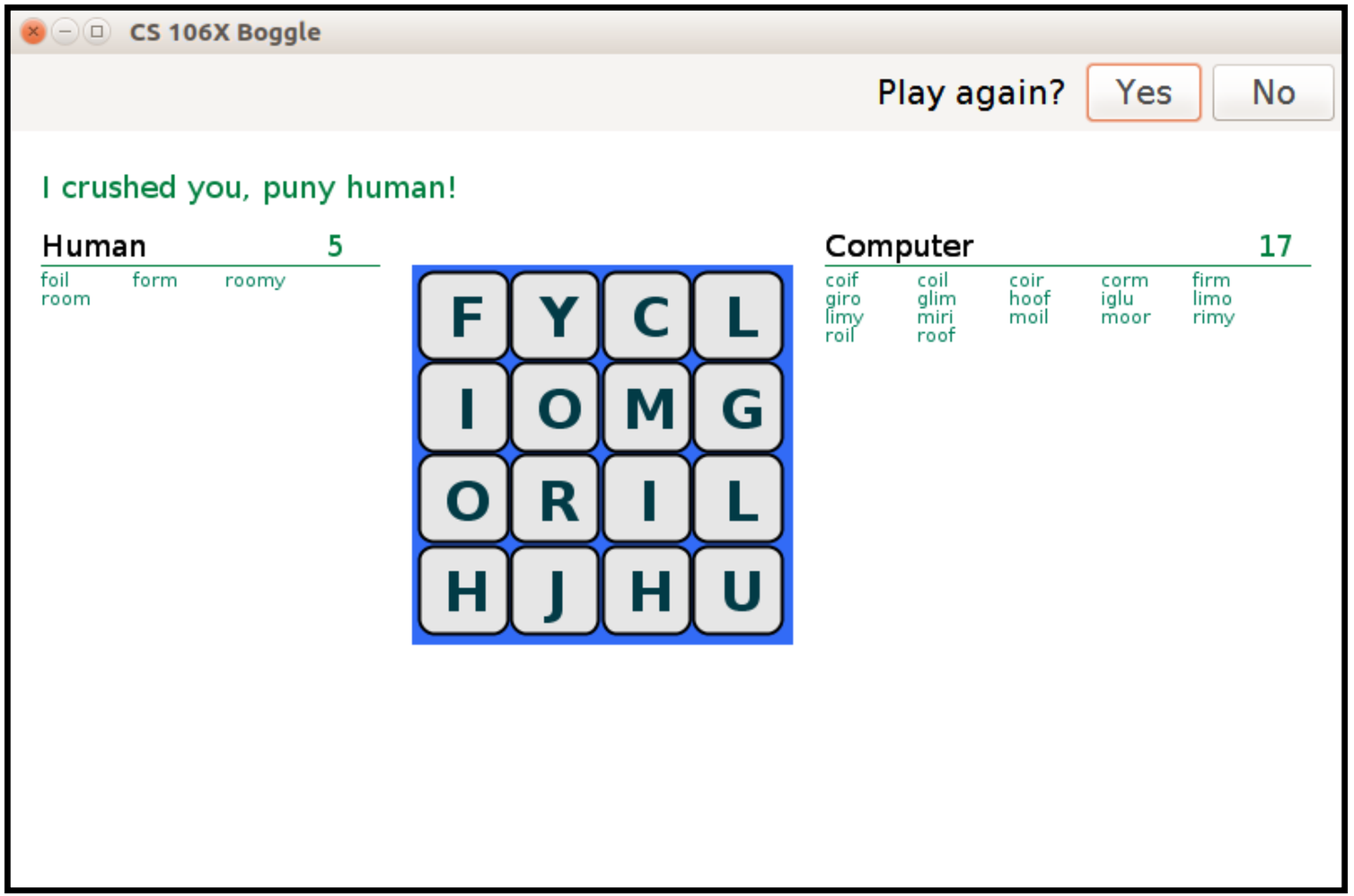


... Order-6



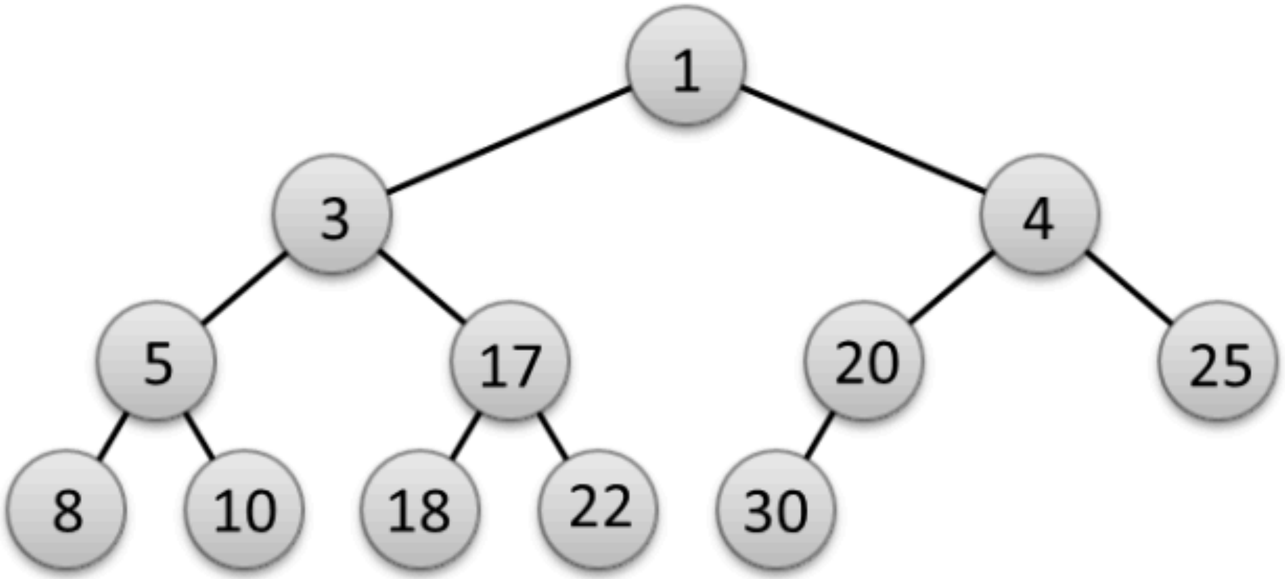
Order-5 tree fractal

Where We Have Been: Backtracking

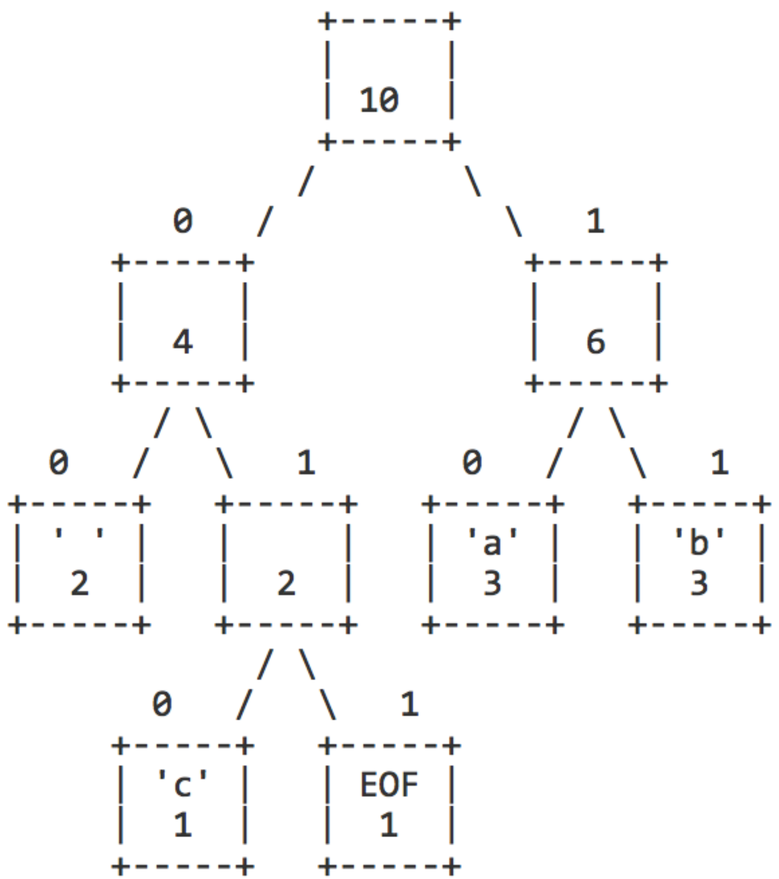


Where We Have Been: Linked Lists and Heaps

zero nodes	`front /`
one node	`front --> | ? | / |`
N nodes	`front --> | ? | --> | ? | --> ... | ? | --> | ? | / |`

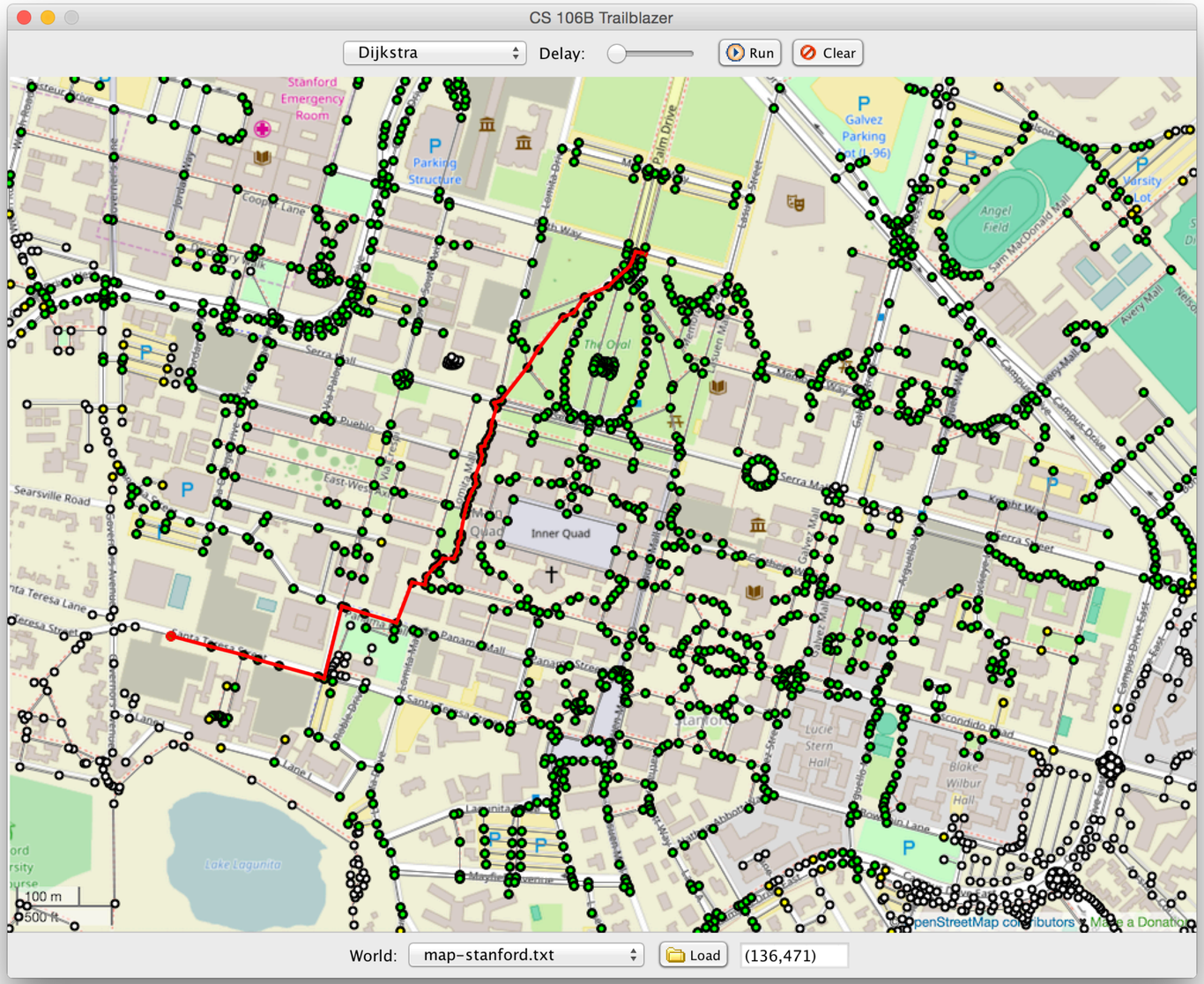


Where We Have Been: Binary Trees



Huffman!

Where We Have Been: Graphs



Where We Have Been: Sorting

So many ways to sort things!

We learned:

- Insertion sort
- Selection Sort
- Merge Sort
- Quicksort

Other sorts:

- Shell Sort
- Heap Sort
- Tim Sort
- Radix Sort
- Bubble Sort

Trying to achieve O(n log n)

Where We Have Been: C++

For many of you, a new language!

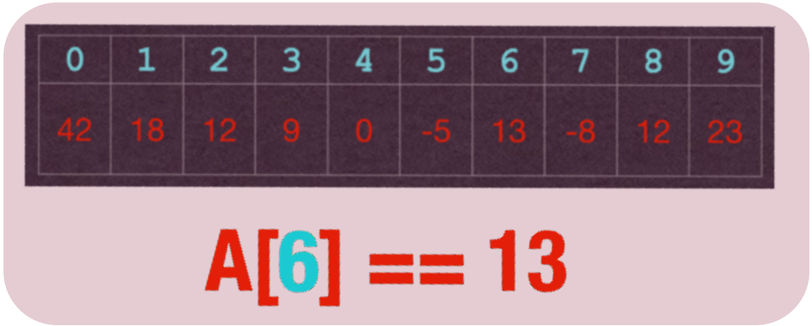
Highlights:

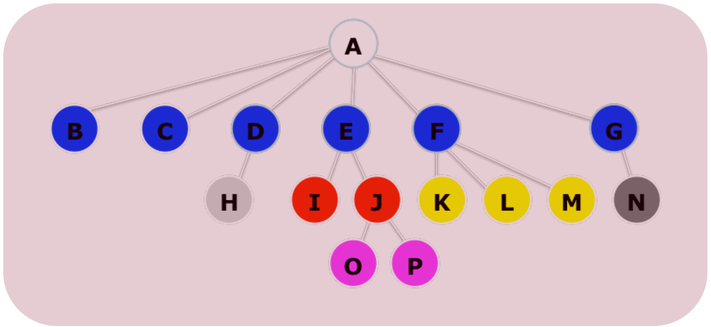
- Object oriented language with classes
- Fast (except our wonky graphics...)
- Extremely robust (too much sometimes)
- Widely used in industry and for making games

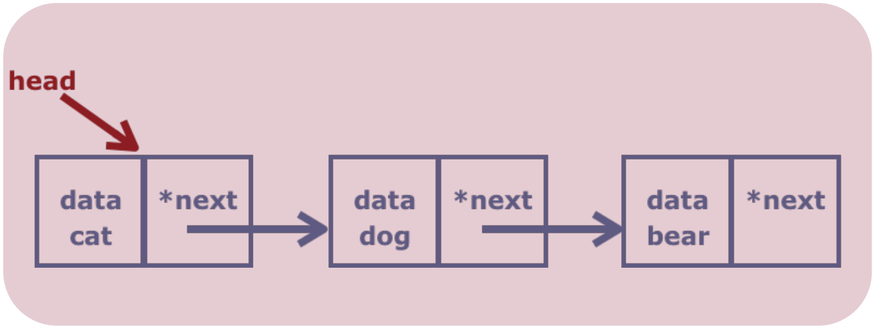
Differences you probably saw from other languages:

- Mutable strings
- Input / Output streams
- Operator overloading
- Pointers
- Memory Management: new, delete
- Inheritance and Polymorphism

The Importance of Data Structures







Why Data Structures are Important

One reason we care about data structures is, quite simply, time. Let's say we have a program that does the following (and times the results):

- Creates four "list-like" containers for data.
- Adds 100,000 elements to each container – specifically, the even integers between 0 and 198,998 (sound familiar?).
- Searches for 100,000 elements (all integers 0-100,000)
- Attempts to delete 100,000 elements (integers from 0-100,000)

What are the results?

The Importance of Data Structures

Structure	Overall(s)
Unsorted Vector	
Linked List	
Hash Table	
Binary Tree	
Sorted Vector	

The Importance of Data Structures

Results:

Structure	Overall(s)
Unsorted Vector	15.057
Linked List	92.202
Hash Table	0.145
Binary Tree	0.164
Sorted Vector	1.563

A factor of 103x

A factor of 636x!

Processor: 2.8GHz Intel Core i7 (Macbook Pro)
Compiler: clang++

Overall, the Hash Table "won" — but (as we shall see!) while this is generally a *great* data structure, there are trade-offs to using it.

Note: In general, for this test, we used optimized library data structures (from the "standard template library") where appropriate. The Stanford libraries are not optimized.

Full Results

Structure	Overall(s)	Insert(s)	Search(s)	Delete(s)
Unsorted Vector	15.057	0.007	10.307	4.740
Linked List	92.202	0.025	46.436	45.729
Hash Table	0.145	0.135	0.002	0.008
Binary Tree	0.164	0.133	0.010	0.0208
Sorted Vector	1.563	0.024	0.006	1.534

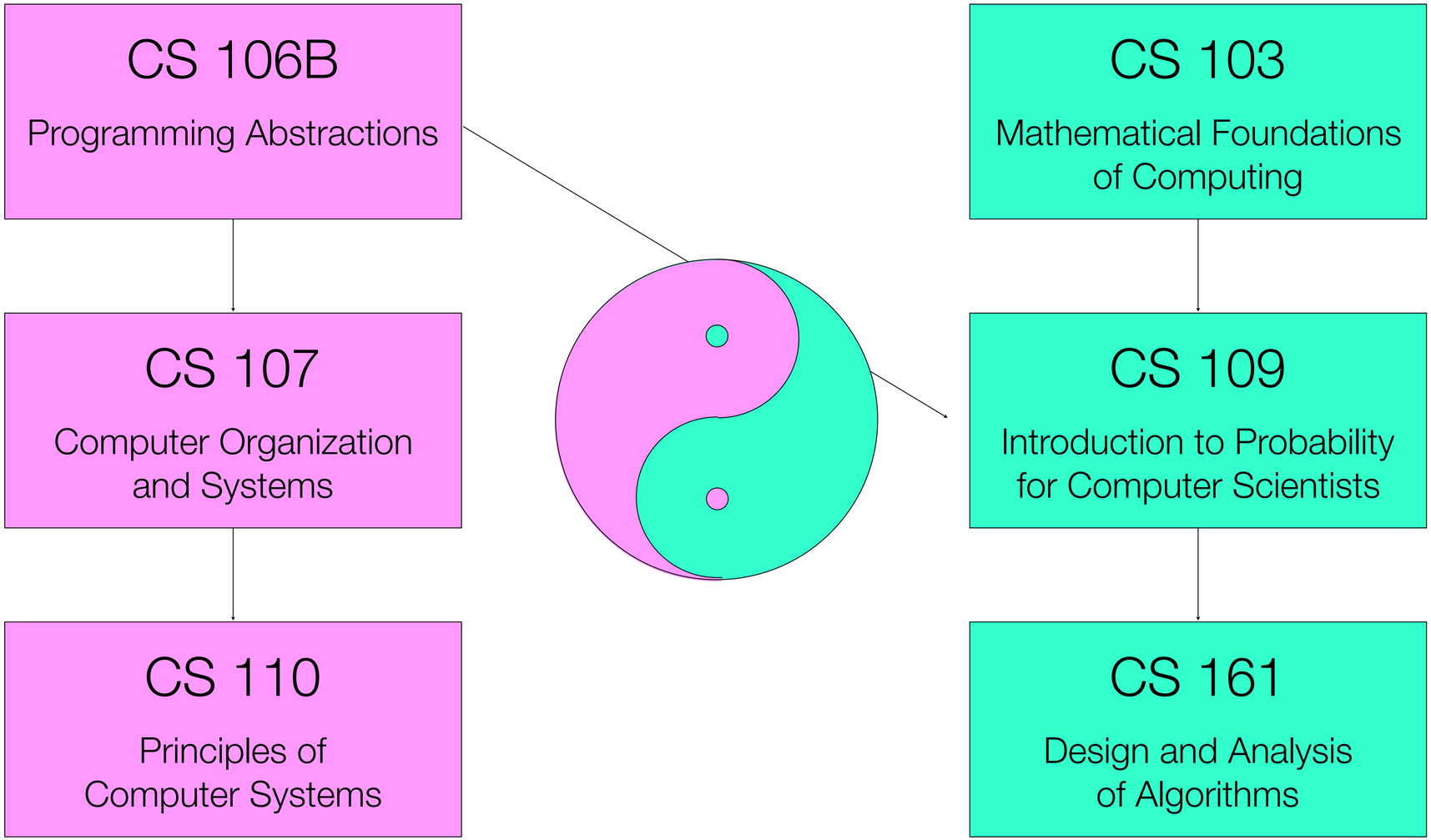
Why are there such discrepancies??

Bottom line:

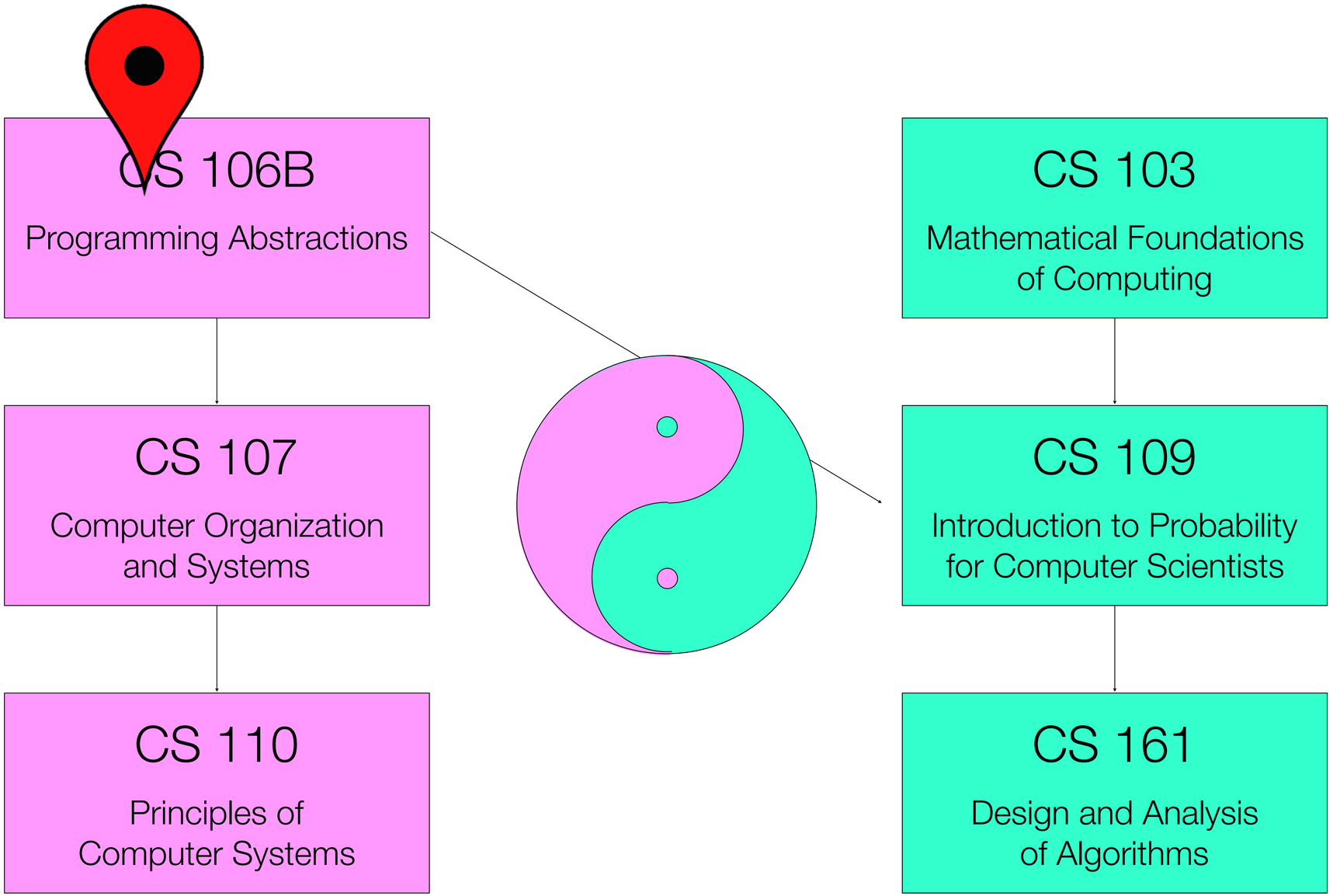
- **Some structures carry more *information* simply because of their design.**
- **Manipulating structures takes time**

Where to from here?

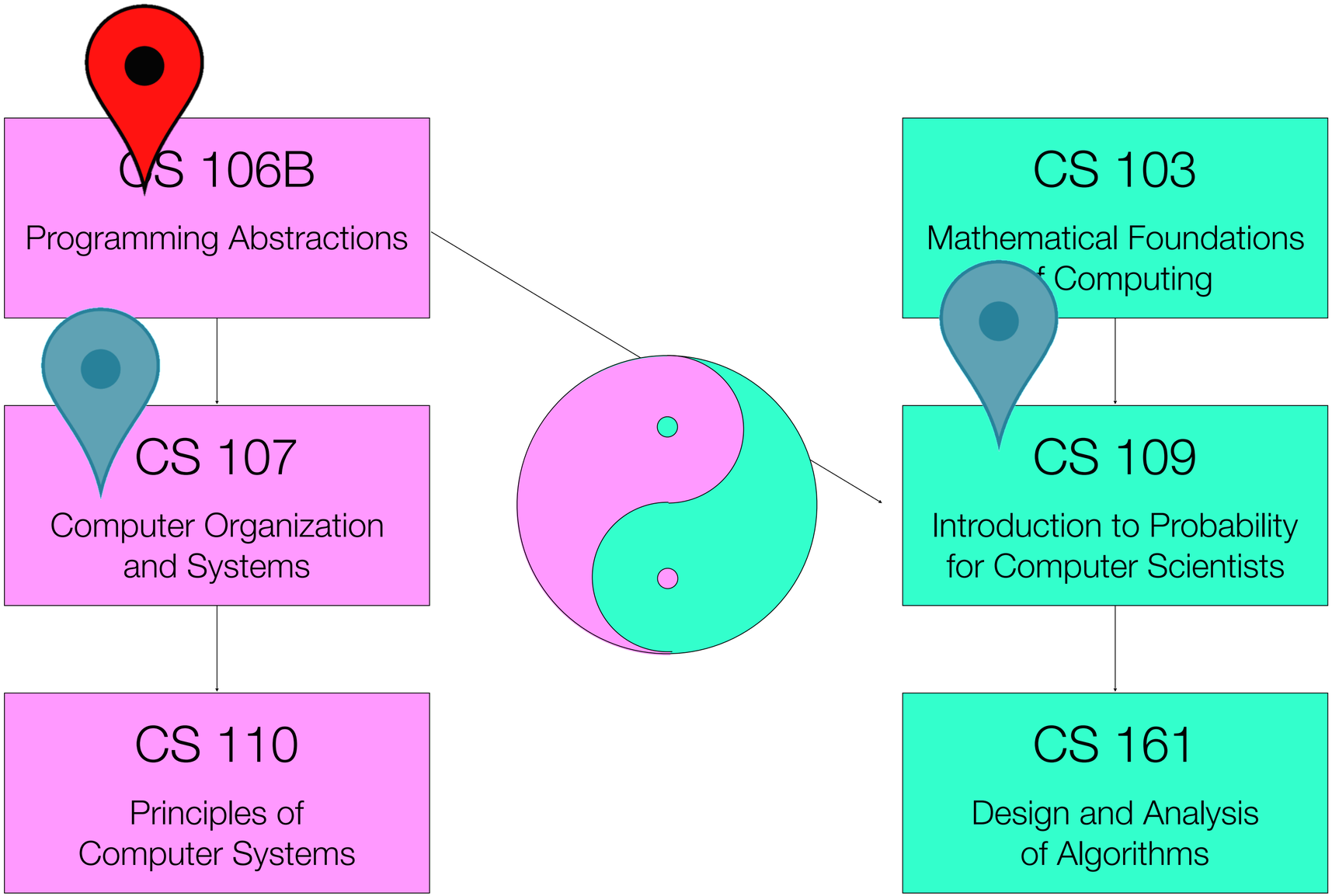
CS Core


CS 106B
Programming Abstractions
CS 107
Computer Organization and Systems
CS 110
Principles of Computer Systems
CS 103
Mathematical Foundations of Computing
CS 109
Introduction to Probability for Computer Scientists
CS 161
Design and Analysis of Algorithms

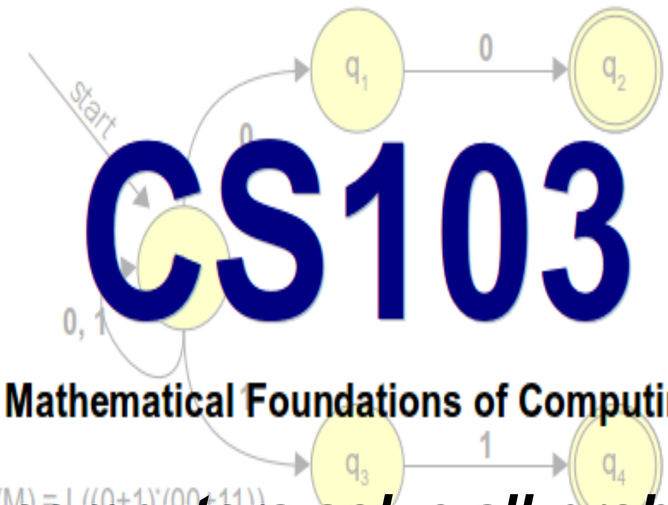
CS Core


CS 106B
Programming Abstractions
CS 103
Mathematical Foundations of Computing
CS 107
Computer Organization and Systems
CS 109
Introduction to Probability for Computer Scientists
CS 110
Principles of Computer Systems
CS 161
Design and Analysis of Algorithms

CS Core


CS 106B
Programming Abstractions
CS 107
Computer Organization and Systems
CS 110
Principles of Computer Systems
CS 103
Mathematical Foundations
f Computing
CS 109
Introduction to Probability for Computer Scientists
CS 161
Design and Analysis of Algorithms

CS 103



CS103

Mathematical Foundations of Computing

Can computers solve all problems?

Spoiler alert: no!

Why are some problems harder than others?

We can do find in an unsorted array in O(N), and we can sort an unsorted array in O(NlogN). Is sorting just inherently a harder problem, or are there better O(N) sorting algorithm yet to be discovered?

How can we be certain about this?

CS107 (kind of like CS106C)

How do we encode text, numbers, programs, etc. using just 0s and 1s?

Where does memory come from? How is it managed?

How do compilers, debuggers, etc. work?

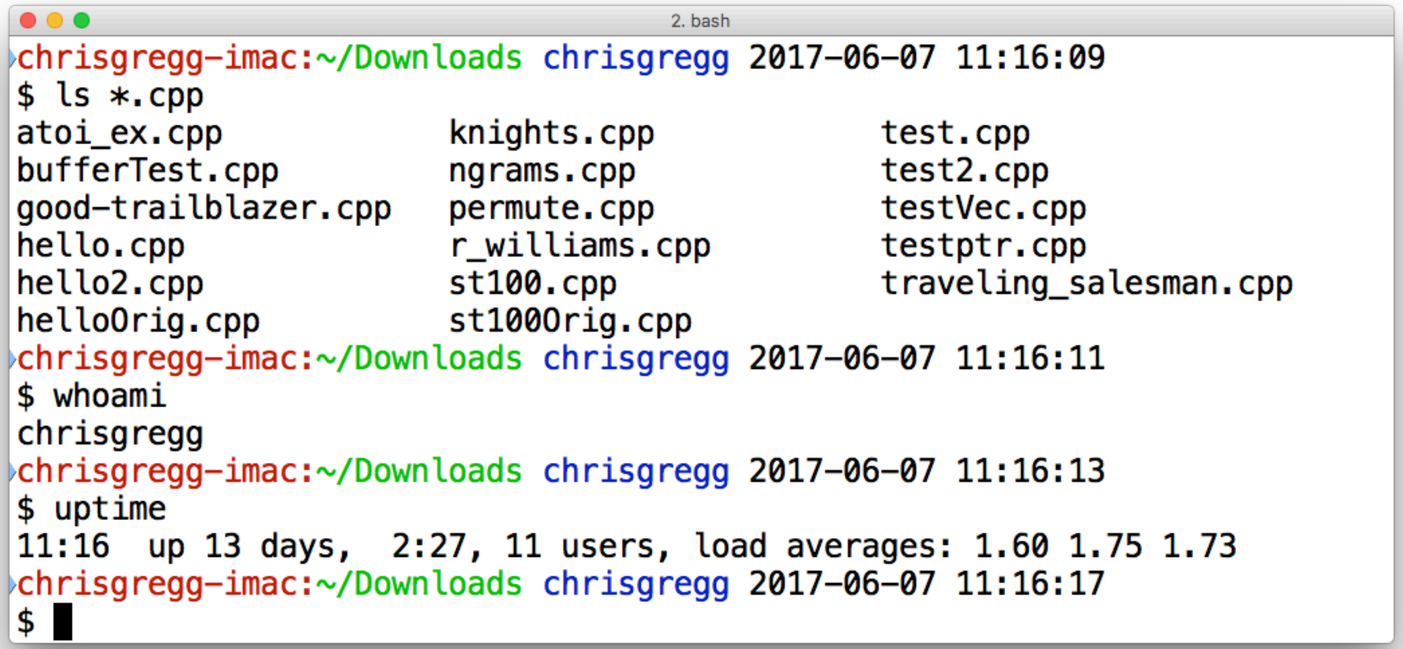
CS107 is *not*

- CS107 is *not* a litmus test for whether you can be a computer scientist.
  - You can be a *great* computer scientist without enjoying low-level systems programming.

- CS107 is *not* indicative of what programming is “really like.”
  - CS107 does a lot of low-level programming. You don't have to do low-level programming to be a good computer scientist.

How to Prepare for cs107

- You **are** prepared! cs106b is a great preparation!
- However...you may want to learn a few things before class
  - Terminal

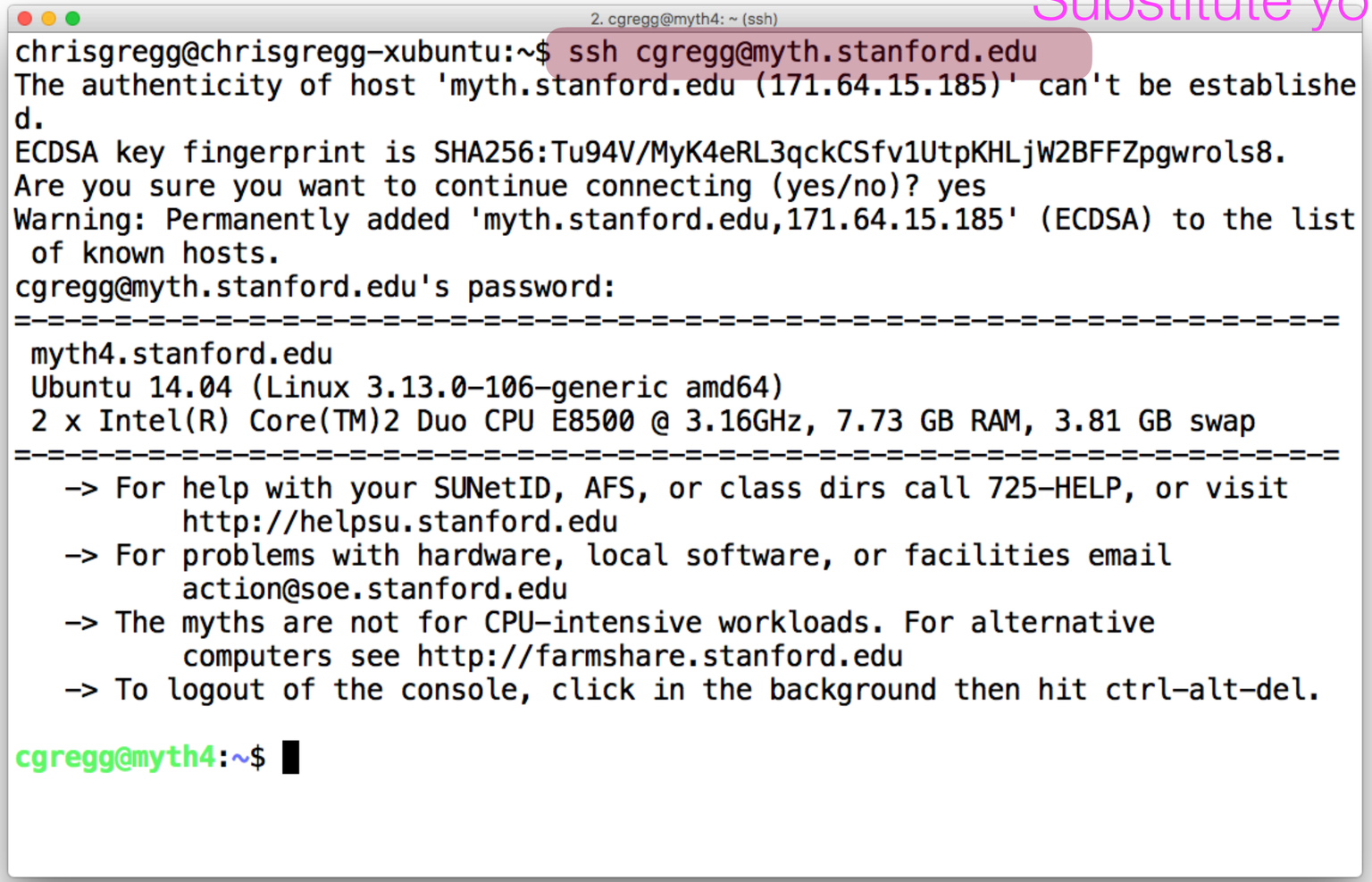



```
/afs/ir/users/c/g/cgregg/cowsay/bin/cowsay Moooo
ls /afs/ir/users/c/g/cgregg/cowsay/share/cows
/afs/ir/users/c/g/cgregg/cowsay/bin/cowsay -f stegosaurus nowr
```

How to Prepare for cs107

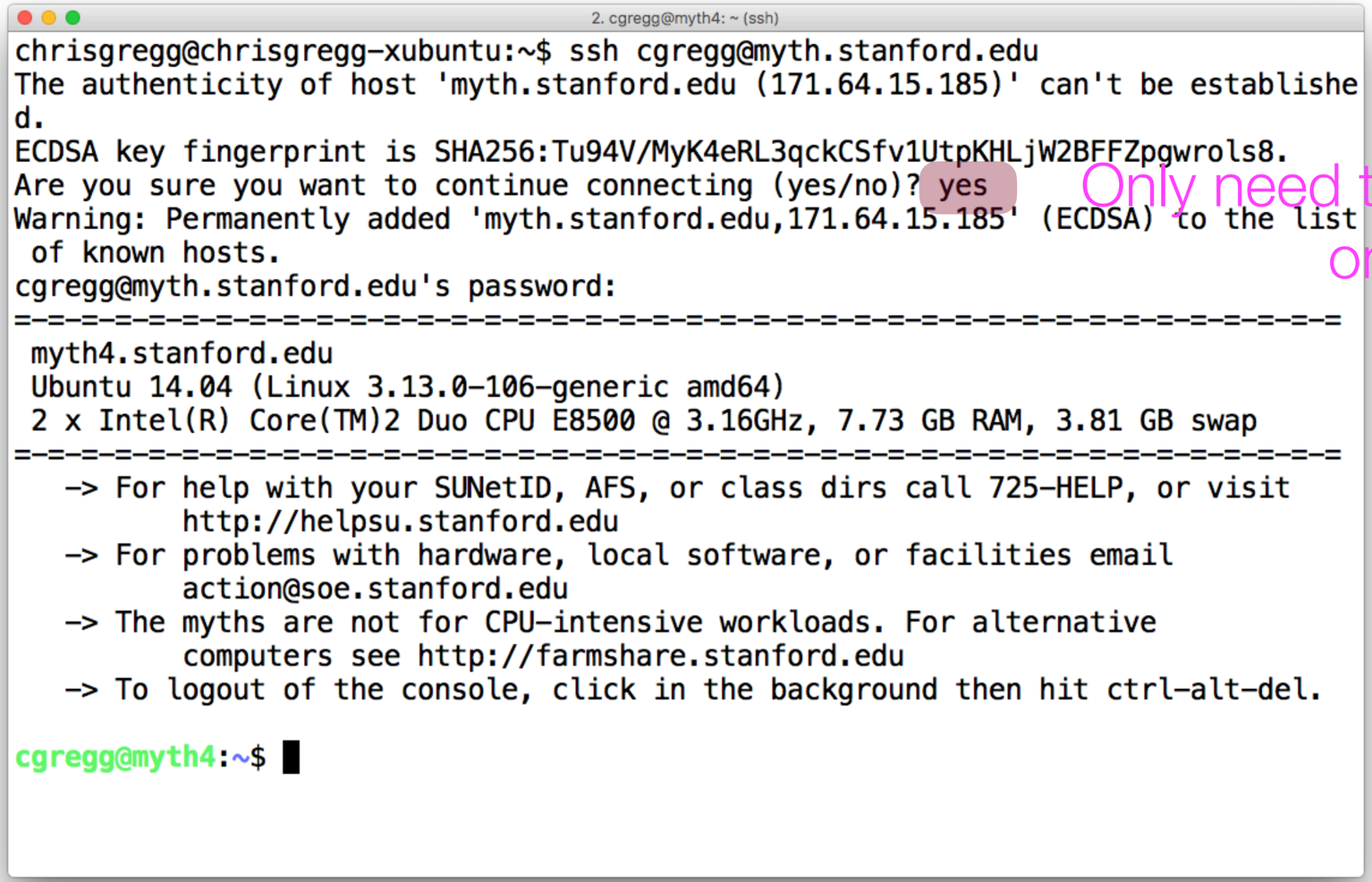
- How to log into Myth: Open Terminal (on Mac / Linux), use MobaXTerm on Windows. Then:

Substitute your SUNet id



How to Prepare for cs107

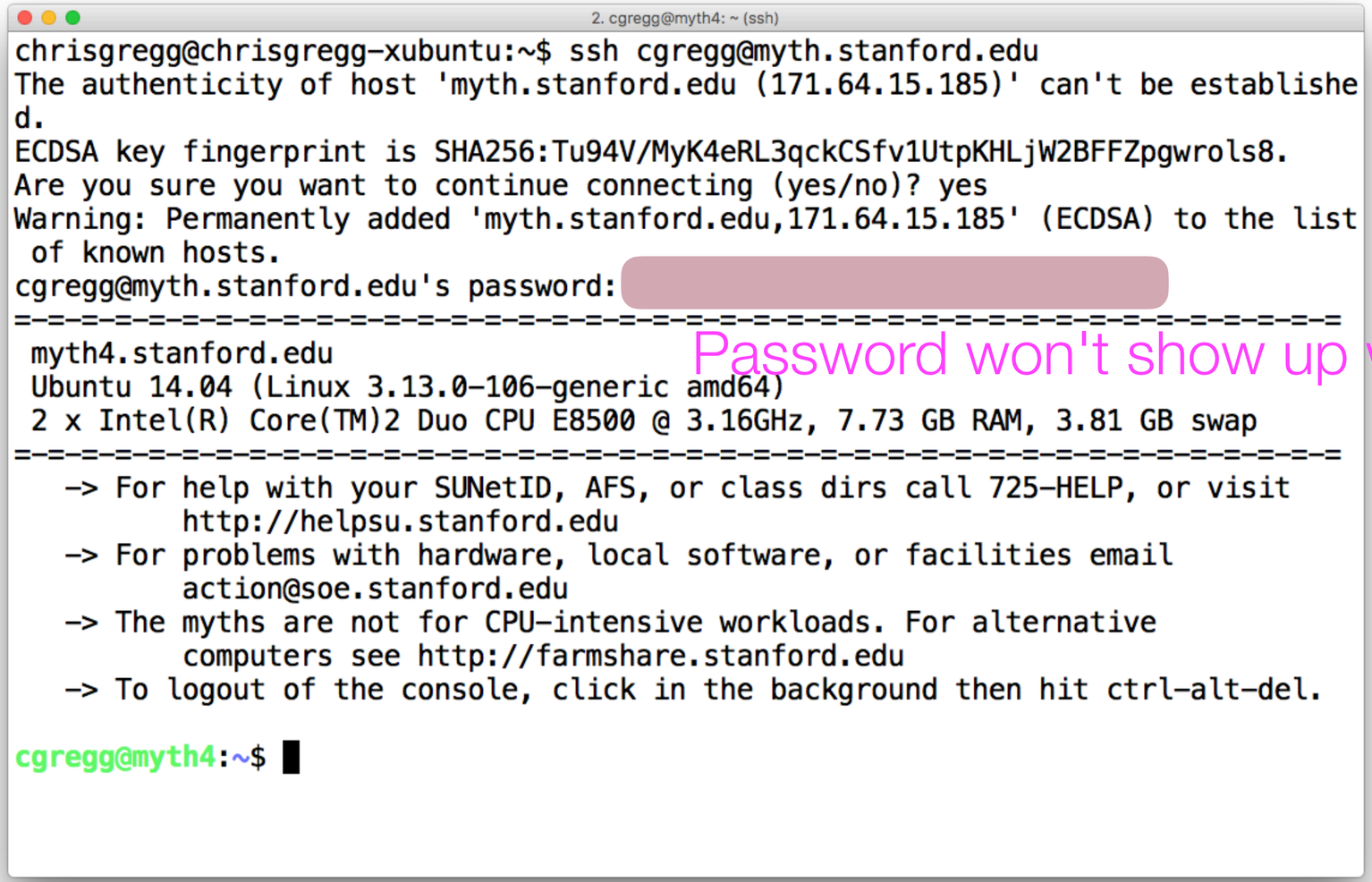
- How to log into Myth: Open Terminal (on Mac / Linux), use MobaXTerm on Windows. Then:



Only need to answer this once.

How to Prepare for cs107

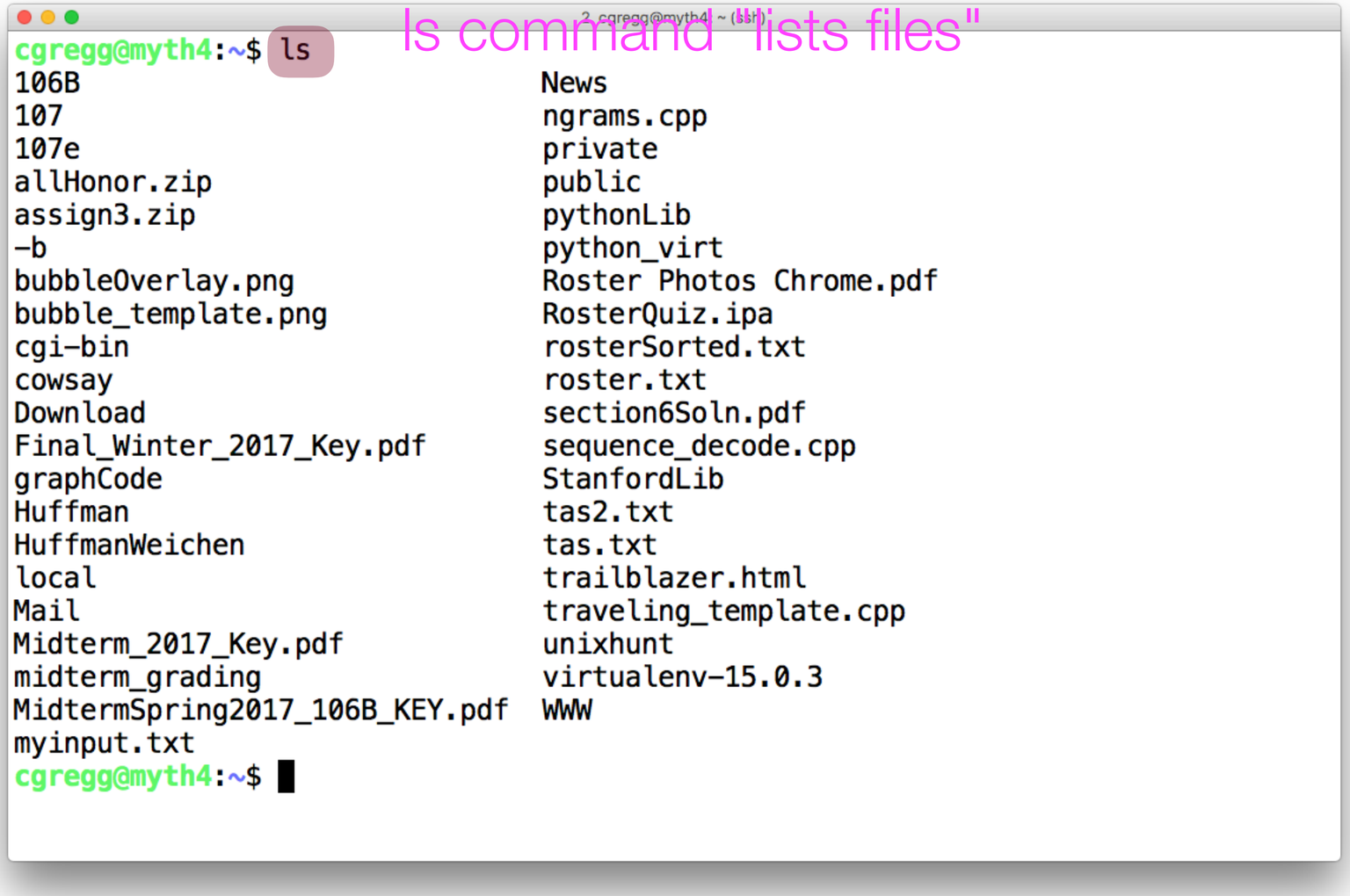
- How to log into Myth: Open Terminal (on Mac / Linux), use MobaXTerm on Windows. Then:



Password won't show up when you type it.

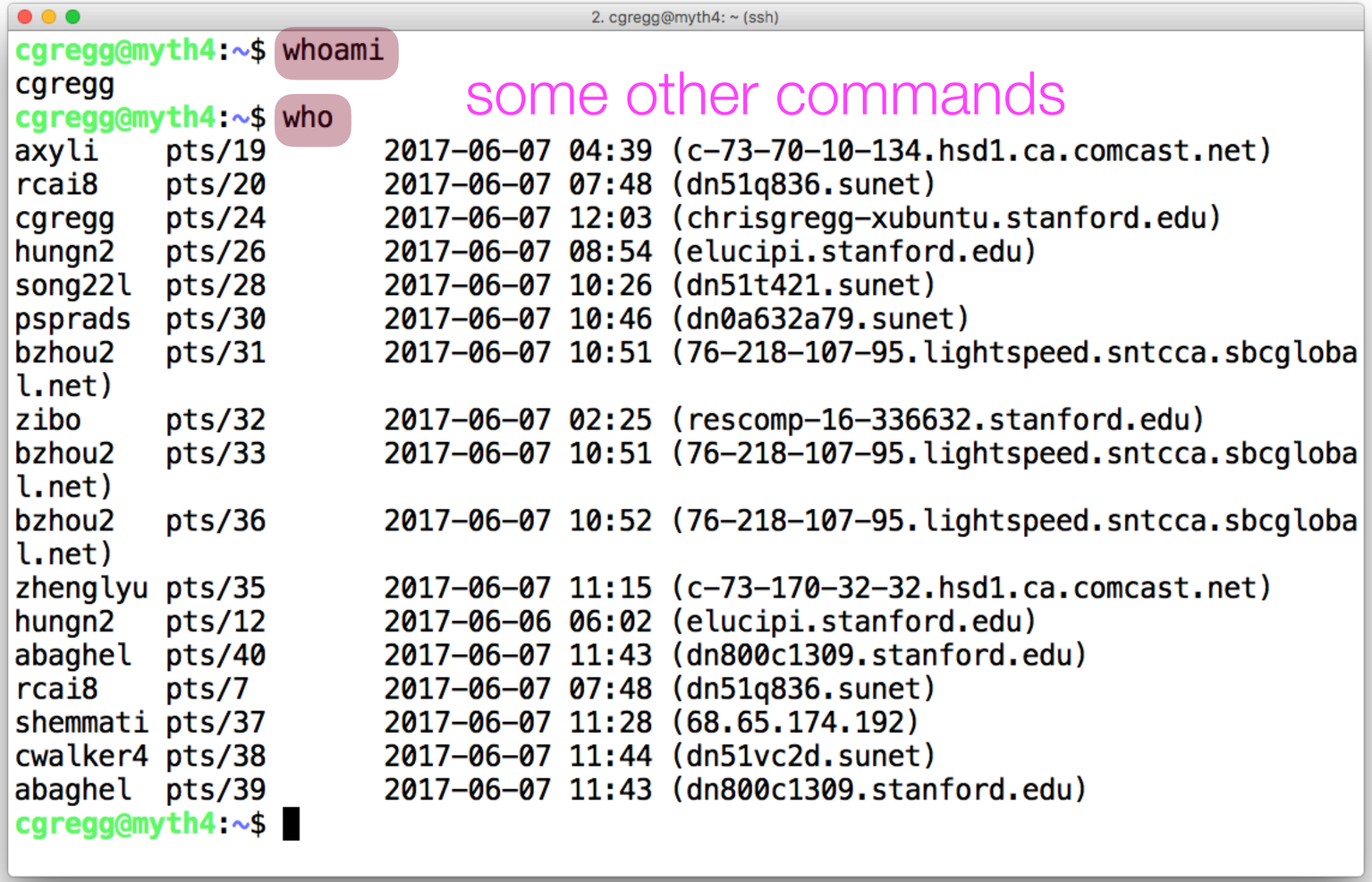
How to Prepare for cs107

- How to log into Myth: Open Terminal (on Mac / Linux), use MobaXTerm on Windows. Then:



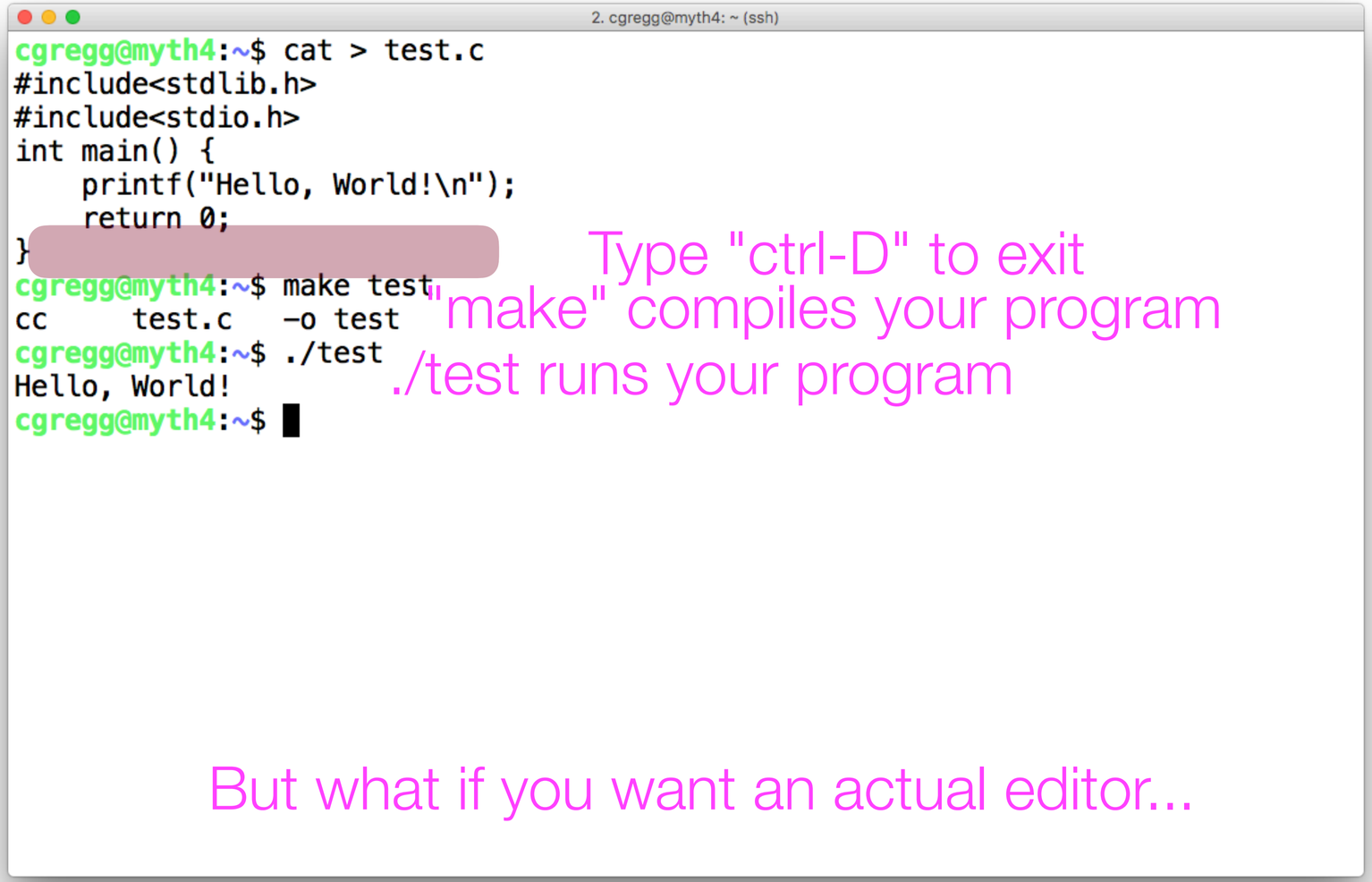
How to Prepare for cs107

- How to log into Myth: Open Terminal (on Mac / Linux), use MobaXTerm on Windows. Then:



How to Prepare for cs107

- Your first C program!



But what if you want an actual editor...

How to Prepare for cs107

- Lots of choices:
  - vi / vim
  - emacs
  - nano
  - Sublime Text from your own computer (slower, must transfer files)

How to Prepare for cs107

Editor war

From Wikipedia, the free encyclopedia

For a type of conflict between Wikipedia editors, see Wikipedia:Edit war

Editor war is the common name for the rivalry between users of the Emacs and vi (Vim) text editors. The rivalry has become a lasting part of hacker culture and the free software community.

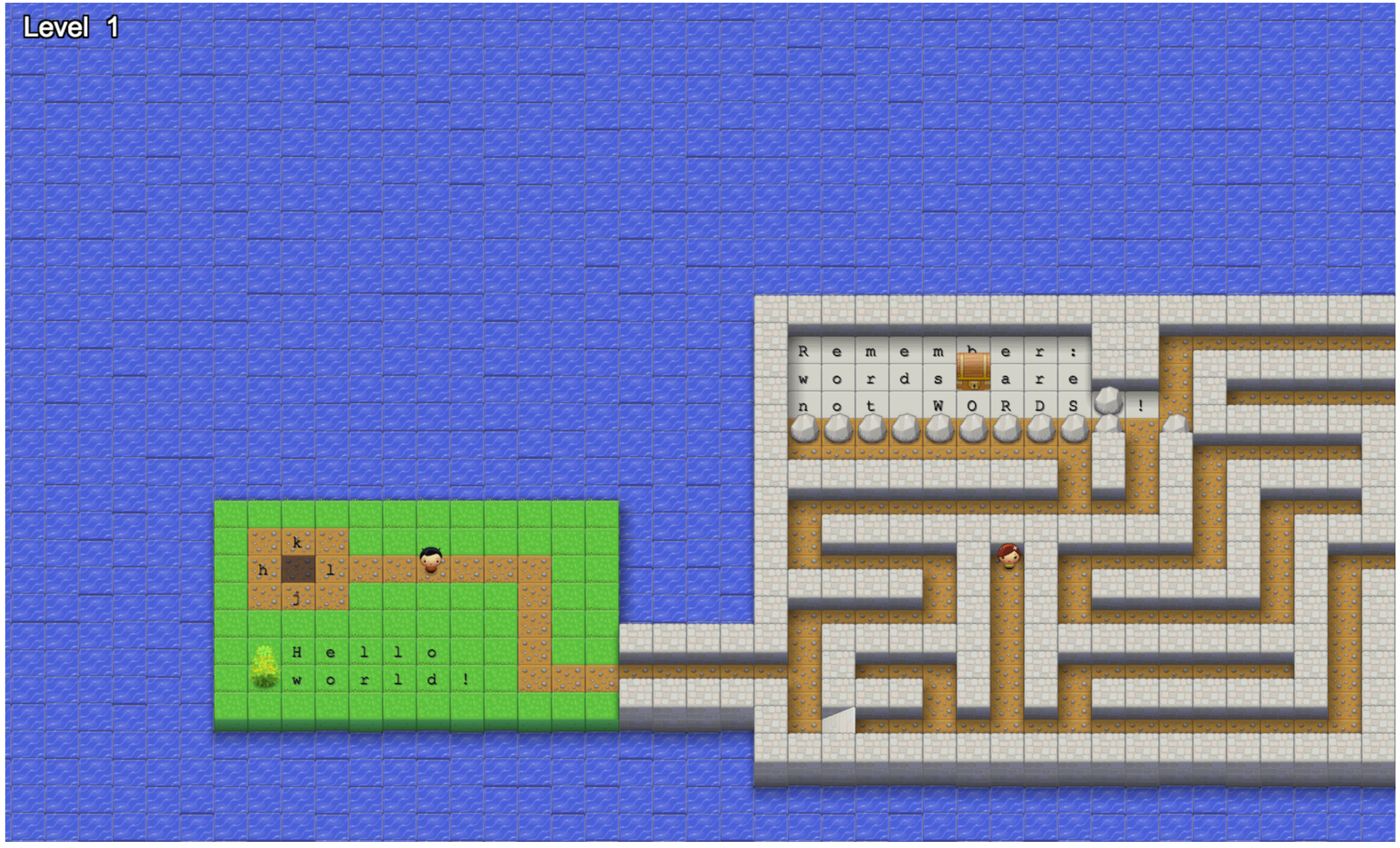
The Emacs vs vi debate was one of the original "holy wars" conducted on Usenet groups,[1] with many flame wars fought between those insisting that their editor of choice is the paragon of editing perfection, and insulting the other, since at least 1985.[2] Related battles have been fought over operating systems, programming languages, version control systems, and even source code indent style.[3][4][5] Notably, unlike other wars (ie, UNIX vs ITS vs VMS, C vs Pascal vs Fortran), the editor war has yet to be resolved with a clear winner, and the hacker community remains split roughly 50/50.[2]

Differences between vi and Emacs [edit]

The most important differences between vi and Emacs are presented in the following table:

How to Prepare for cs107

- Vim Adventures: https://vim-adventures.com/



How to Prepare for cs107

- You might want to learn more about pointers, and pointers-to-pointers... https://www.tutorialspoint.com/cprogramming/c_pointer_to_pointer.htm

```c
#include <stdio.h>

int main () {

   int  var;
   int  *ptr;
   int  **pptr;

   var = 3000;

   /* take the address of var */
   ptr = &var;

   /* take the address of ptr using address of operator & */
   pptr = &ptr;

   /* take the value using pptr */
   printf("Value of var = %d\n", var );
   printf("Value available at *ptr = %d\n", *ptr );
   printf("Value available at **pptr = %d\n", **pptr);

   return 0;
}
```

How to Prepare for cs107

- What is this "printf()" function??
- It is the "cout" for C

Let's talk about some C differences...

How to Prepare for cs107

Going from C++ to C: *"C has the speed and efficiency of assembly language combined with readability of assembly language."*

- You will find C to be similar in feel to C++ (C++ is based on C, after all)
- Things that are the same:
  - built-in data types: int, char, float, double, long, unsigned int
  - array indexing (using brackets, e.g., v[4])
  - function declarations
  - modularity between .h and .c files
  - flow: while, if, for, etc.

How to Prepare for cs107

Going from C++ to C

- Things that are different:
  - There isn't a "string" type. Strings are null terminated arrays of chars.
  - There are no classes, no objects, and no constructors/ destructors.
  - Memory management is more involved than in C++
    - To allocate memory, C uses "malloc()" (so can C++, but normally we use `new()`)

How to Prepare for cs107

C strings: null terminated char arrays

- There is no string class in C. Let me repeat: there is no string class in C.
- Strings are simply arrays of characters, with the final character of the array set aside for the NULL character.
- You must be careful to avoid buffer overflows when dealing with char strings.

How to Prepare for cs107

Going from C++ to C

- Things that are the same but that you might not have learned about yet:
  - There are entities called “void pointers” which can point to *any* type
  - You often need to cast variables to different types
  - “Pointer arithmetic” is often used instead of bracket notation

How to Prepare for cs107

Going from C++ to C

- **`malloc()`** and **`free()`**, and **`sizeof()`**: used for memory management instead of new and delete

- **`malloc()`** is used to reserve memory from the heap. You must pass it the size in bytes of the amount of memory you want. How do you know the size in bytes? You use sizeof():

- **`sizeof(int)`** returns the size of an integer. On our machines, this is 4 bytes, or 32-bits.

How to Prepare for cs107

Going from C++ to C

- `malloc()` returns a pointer to memory, e.g.,

```
/* an array of 10 ints from the heap */
int *int_ptr;
int_ptr = malloc(sizeof(int)*10);
```

- `malloc()` returns `NULL` if the system runs out of memory.

How to Prepare for cs107

Going from C++ to C

- `free()` simply frees the previously allocated memory:

```
/* an array of 10 ints from the heap */
int *int_ptr;
int_ptr = malloc(sizeof(int)*10);
free(int_ptr);
```

How to Prepare for cs107

Going from C++ to C

- Pointer arithmetic is really important in C, and you have to understand it.

```
int main() {
      /* an array of 10 ints */
      int *int_ptr;
      int_ptr = (int *)malloc(sizeof(int)*10);

      int i;
      for(i=0;i<10;i++) {
         *(int_ptr+i)=i*10;
      }
      for(i=0;i<5;i++) {
         printf("%d\n",*(int_ptr+i*2));
      }
      free(int_ptr);
         return 0;
   }
```

What will this print out?
Oops — we'd better take a detour into printf() land.

How to Prepare for cs107

`printf()` — *the* way to print to stdout in C

- `printf()` writes a formatted string to the standard output. The format can include *format specifiers* that begin with % and additional arguments are inserted into the string, replacing their formatters.

- Example: print the int variable, d:

```
int i = 22;
printf("This is i: %d\n",i);


output:
This is i: 22
```

How to Prepare for cs107

`printf()` — *the* way to print to stdout in C

- lots of specifiers

`o`	Unsigned octal	`610`
`x`	Unsigned hexadecimal integer	`7fa`
`X`	Unsigned hexadecimal integer (uppercase)	`7FA`
`f`	Decimal floating point, lowercase	`392.65`
`F`	Decimal floating point, uppercase	`392.65`
`e`	Scientific notation (mantissa/exponent), lowercase	`3.9265e+2`
`E`	Scientific notation (mantissa/exponent), uppercase	`3.9265E+2`
`g`	Use the shortest representation: `%e` or `%f`	`392.65`
`G`	Use the shortest representation: `%E` or `%F`	`392.65`
`a`	Hexadecimal floating point, lowercase	`-0xc.90fe`
`A`	Hexadecimal floating point, uppercase	`-0XC.90FE`
`c`	Character	`a`
`s`	String of characters	`sample`
`p`	Pointer address	`b8000000`
`n`	Nothing printed. The corresponding argument must be a pointer to a `signed int`. The number of characters written so far is stored in the pointed location.	
`%`	A `%` followed by another `%` character will write a single `%` to the stream.	`%`

How to Prepare for cs107

`printf()` — more examples

```
#include <stdio.h>
int main()
{
   printf ("Characters: %c %c \n", 'a', 65);   Characters: a A
   printf ("Decimals: %d %ld\n", 1977, 650000L);   Decimals: 1977 650000
   printf ("Preceding with blanks: %10d \n", 1977);   Preceding with blanks:       1977
   printf ("Preceding with zeros: %010d \n", 1977);   Preceding with zeros: 0000001977
   printf ("Radices: %d %x %o %#x %#o \n", 100, 100, 100, 100, 100);   Radices: 100 64 144 0x64 0144
   printf ("floats: %4.2f %+.0e %E \n", 3.1416, 3.1416, 3.1416);  floats: 3.14 +3e+000 3.141600E+000
   printf ("Width trick: %*d \n", 5, 10);  Width trick:    10
   printf ("%s \n", "A string");  A string
   return 0;
}
```

How to Prepare for cs107

Going from C++ to C (back to our example)

- Pointer arithmetic is really important in C, and you have to understand it.

```
int main() {
        /* an array of 10 ints */
        int *int_ptr;
        int_ptr = (int *)malloc(sizeof(int)*10);

        int i;
        for(i=0;i<10;i++) {
            *(int_ptr+i)=i*10;
        }
        for(i=0;i<5;i++) {
            printf("%d\n",*(int_ptr+i*2));
        }
        free(int_ptr);
            return 0;
    }
```

What will this print out?

0
20
40
60
80

CS107E


CS107E
Computer Systems from the Ground Up
Pat Hanrahan
Dawson Engler
Julie Zelenski

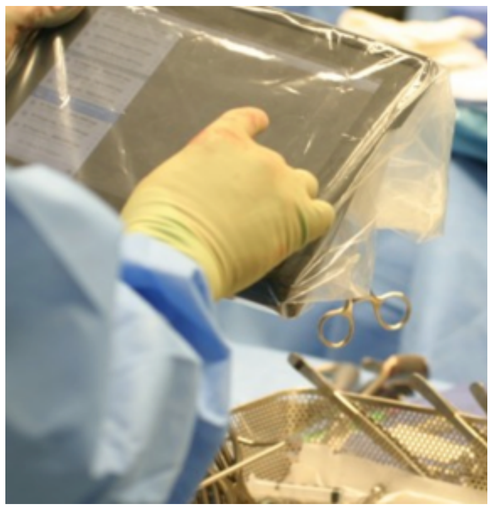
CS109

Foundations of probability

Narrative driven

Intro to Machine Learning

Computer Science Affects Every Field



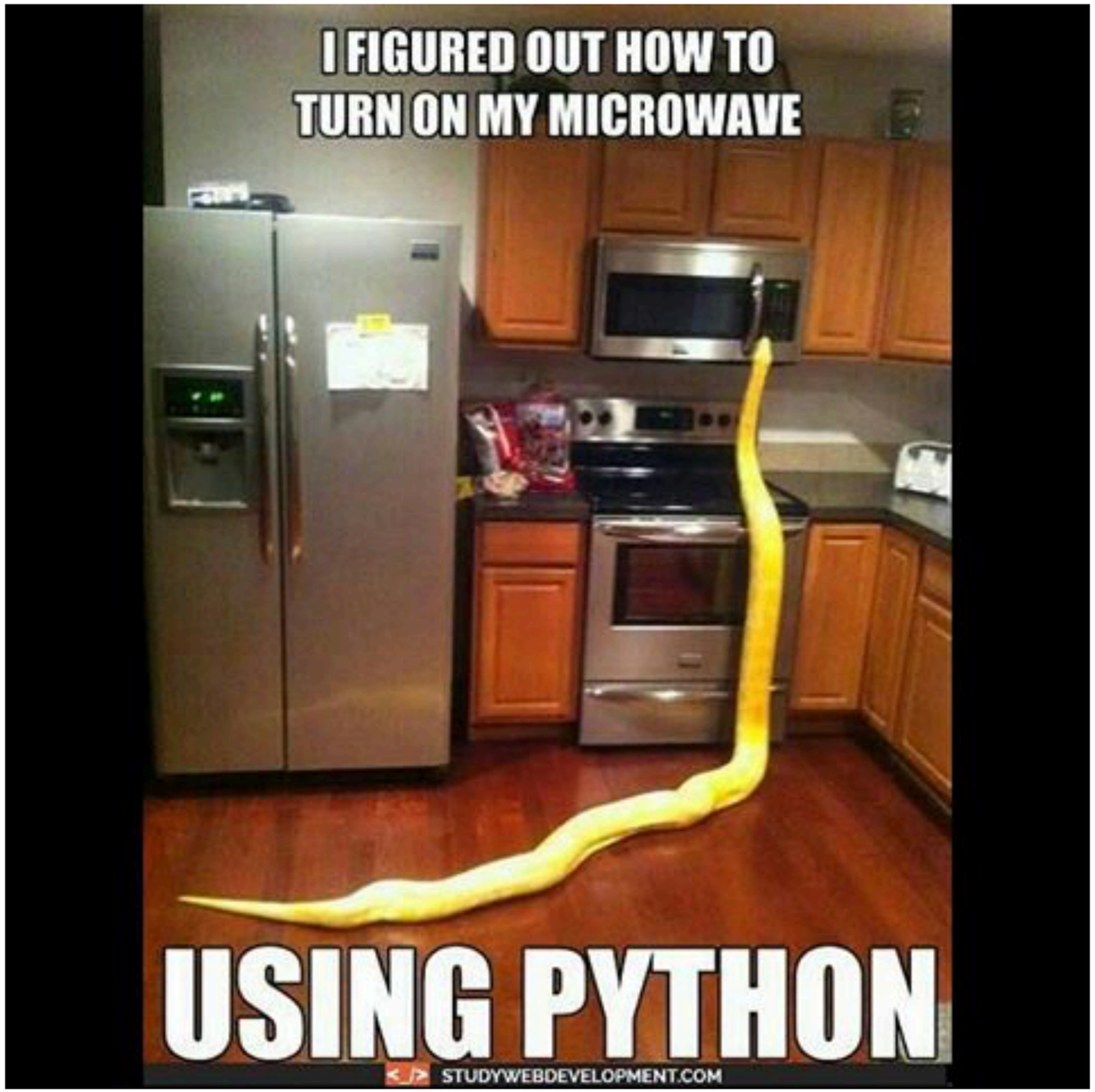


Courses Aren't Necessary!

Things to learn on your own:

- **A new language. Good candidates?**
  - Python: used *everywhere*, easy to learn, easy to write quick programs. Best online resource: https://www.reddit.com/r/Python/ (see right side-bar)
  - Haskell: a "functional" programming language. Best online resource: *Learn You a Haskell for Great Good*
- **iOS / Android Programming: Why not learn how to program your phone?**
  - Best iOS resource: https://www.raywenderlich.com
  - Good tutorials link: http://equallysimple.com/best-android-development-video-tutorials/
  - Want to code for all phones (and the web, and the desktop?) Check out React Native: https://facebook.github.io/react-native/
- **Hardware: Raspberry Pi, Arduino, FPGA: Hardware is awesome!**
  - Raspberry Pi resources: https://www.reddit.com/r/raspberry_pi/
  - Arduino Resources: https://www.reddit.com/r/arduino/
  - FPGA resources: http://www.embedded.com/design/prototyping-and-development/4006429/FPGA-programming-step-by-step
- **GPU and Multicore Programming: hard, but your code can *fly***
  - Your GPU might have hundreds of individual processors. Resources: http://gpgpu.org

Python


I FIGURED OUT HOW TO
TURN ON MY MICROWAVE
USING PYTHON
STUDYWEBDEVELOPMENT.COM

It is the Time and Place for CS

Thank You

Congrats (in advance)

References and Advanced Reading

- **References:**
  - https://en.wikipedia.org/wiki/Theory_of_computation
  - https://explorecourses.stanford.edu/search;jsessionid=1xnekjf0f94bc1gql4to9qrgzp?view=catalog&academicYear=&page=0&q=CS&filter-departmentcode-CS=on&filter-coursestatus-Active=on&filter-term-Autumn=on
  - https://www.theguardian.com/technology/self-driving-cars