

Programming Abstractions in C++

CS106B

Instructors:
Cynthia Bailey
Chris Gregg

Meet your instructors: Chris Gregg

- Undergraduate degree in Electrical Engineering (Johns Hopkins), Master's in Education (Harvard), Ph.D. in Computer Engineering (University of Virginia).
- After undergrad school, I joined the Navy as a Cryptologist. I was on active duty for about seven years, and then in the reserves for another 15 years. I deployed all over the world, and lived in Australia for two years.
- After my active duty service, I became a high school physics teacher, and did that for about seven years in Massachusetts and in Santa Cruz.
- Eventually, I got my Ph.D. and decided to keep teaching, first at Tufts University, and then at Stanford. This is my eighth year at Stanford.
- I love to tinker — my espresso machine is connected to the internet. stop by my office some time to see my typewriter projects.



Meet your instructors: Cynthia Bailey

- This year my partner and I are the new RFs in J-Ro House in Sequoia.
- Last year I was on leave from Stanford, living in DC and advising the U.S. Senate on AI legislation. I met with stakeholders like NATO officials, CEOs, artists, and academics, and wrote bills relating to AI/tech.
- I've been teaching at Stanford for over 10 years, primarily CS106B and CS103. I also teach seminars "AI Governance" and "Race & Gender in Silicon Valley."
- For both undergrad and PhD, I studied CS at UC San Diego, focus on supercomputing and machine learning.
- I've worked as a software engineer or consultant at startups, NASA, and law firms.
- Hobbies: Family, biking, hiking, cooking, "like a cat lady, but for chickens"



Meet your Head TA: Jonathan Coronado

STANFORD BS '24, CURRENTLY MS '25

- I took CS106B Winter 2021!

TEACHING

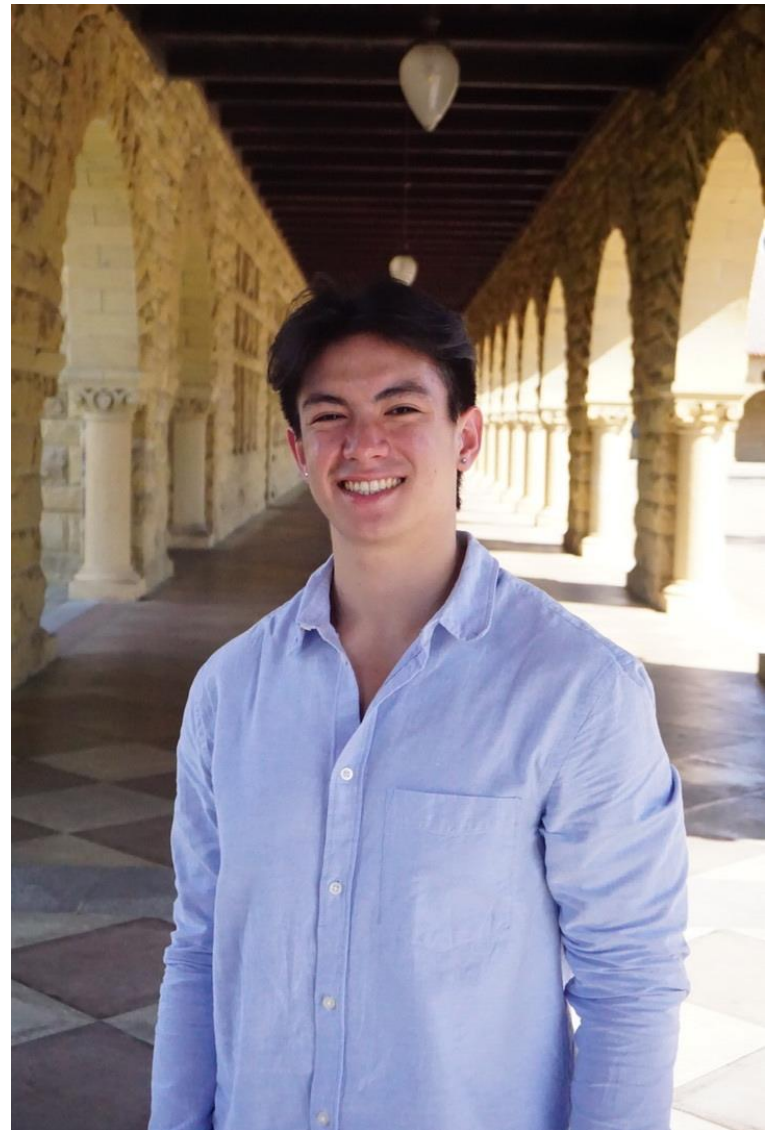
- SL since Fall '22, first time head TA!
- Here to help you succeed in 106B! Please do not hesitate to reach out to me at jonathan.coronado@stanford.edu with anything you need!

SOFTWARE ENGINEERING

- Most recently Backend Engineer at DoorDash, AWS before that

AWAY FROM KEYBOARD

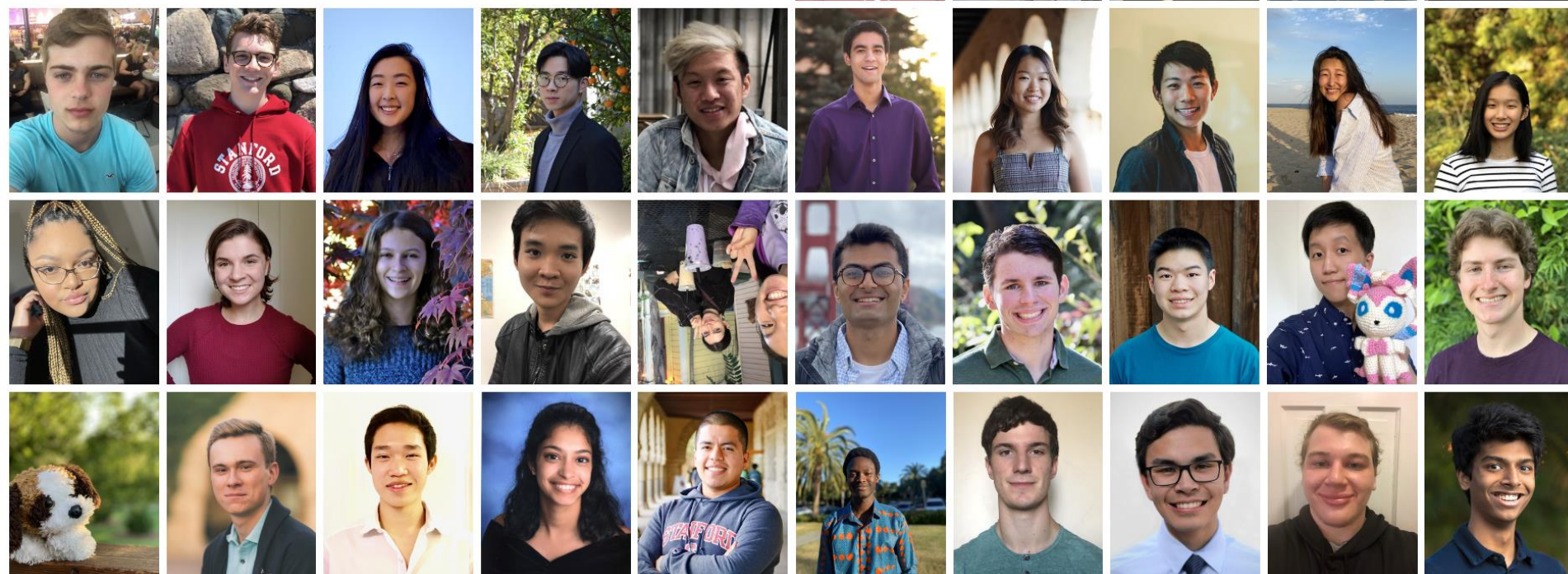
- Running, crosswords, lifting, the seahawks!



Discussion Section, Section Leaders (“SLs”)

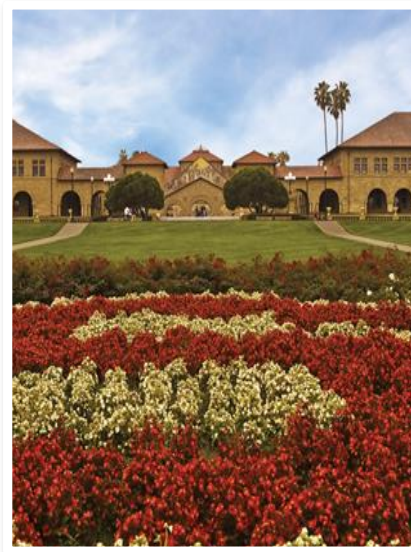
Section Leaders are helpful undergraduate assistants

- Your personal trainer in 106B
- Meet with the same SL each week
- (consider being one in the future!)



Course Logistics

QUICK OVERVIEW OF HOW TO
EARN THE GRADE YOU WANT
IN CS106B

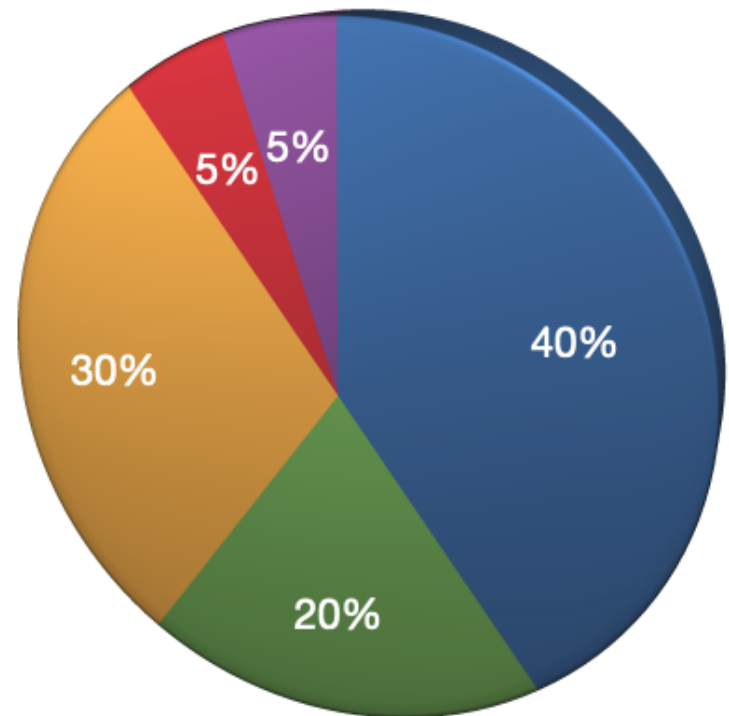
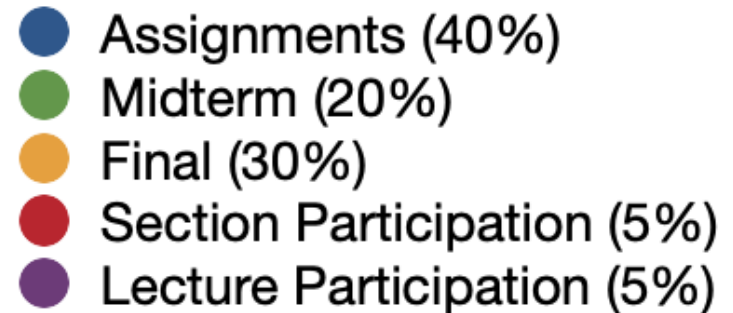


Course Grade Overview

Your course grade will be calculated as follows:

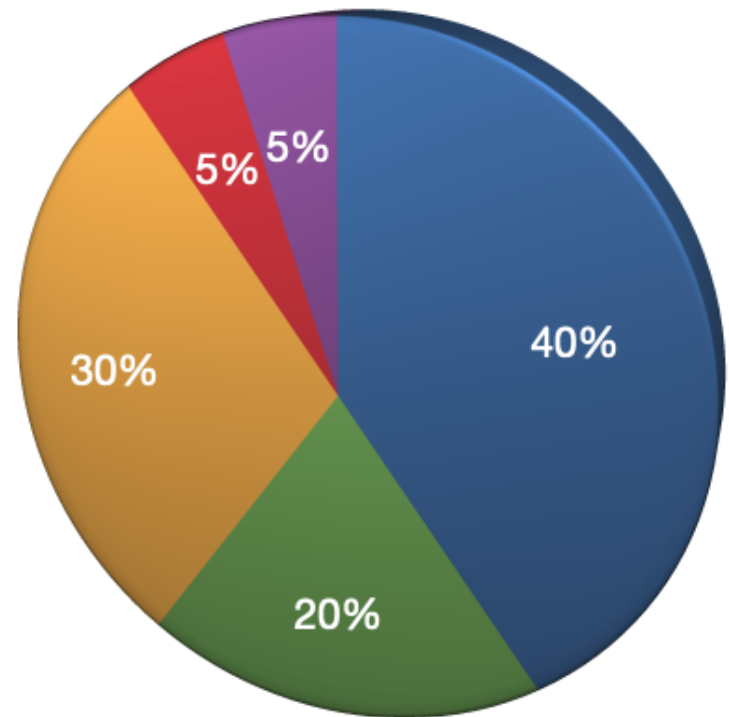
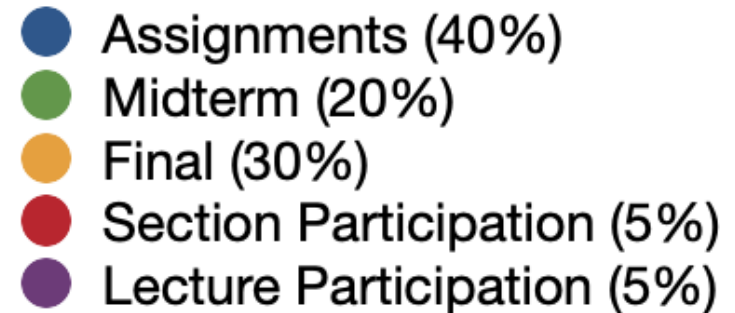
- 40% assignments
- 20% Midterm
- 30% Final
- 5% Section participation
- 5% Lecture participation

(Details and caveats on next slide.)



Course Grade Overview

- The assignments are the most important part of the course, and weighted accordingly.
- We also want section discussions to be robust, so we are including a section participation component.
 - To get an “A” in the class, you may only miss one section.
- **Midterm:** Tuesday, October 29th, 7-9pm PST
- **Final:** Monday, December 9th, 8:30am-11:30am, PST Note:
 - You must pass the final exam in order to pass the course.
- Important: If you are an undergraduate, you must take the course for 5 units. We will not assign grades to undergraduates who take the course for fewer than 5 units.



CS106L: Standard C++ Programming

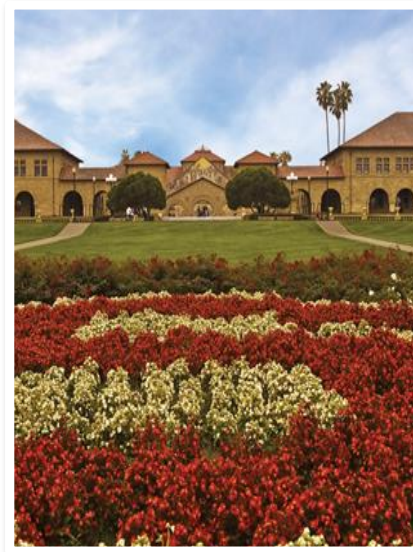
- 1 unit
- T/Th 4:30-5:50pm, Thornton 110
- **Course Info: cs106l.stanford.edu**

- Covers additional topics on C++ features not included in CS106B, where our focus is on the algorithms used to manage large datasets—which apply regardless of programming language used—and not as much on the details of C++ syntax.
 - Templates, streams, lambdas, operator overloading, R-values, and more!

- **Questions?** Reach out to Fabio and Jacob!
(fabioi@stanford.edu & jtrb@stanford.edu)

CS106B Course Culture

HOW TO SUCCEED IN THIS
LONGSTANDING COMMUNITY
AT STANFORD

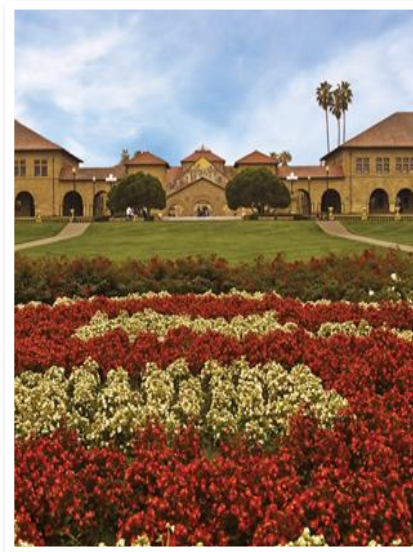


Community norms and expectations

- **Celebrate discovery and growth.** No gatekeeping, shaming, or comparisons based on who knew what coming in.
 - *Example of things we're not going to do: audience "questions" in lecture that are just showing off that you know some jargon.*
- **Shed "zero-sum" and scarcity attitudes.** There are plenty of tech jobs.
 - *Others gaining strength in the power of coding doesn't take power away from you. Be helpful and encouraging, try to feel as genuinely happy when others around you succeed as when you succeed.*
- **Do your own work.** We take this very seriously, because that's how you grow.
 - *Nobody gets good at yoga by watching videos. You have to get on the mat, and sometimes you have to sweat. No shortcuts. We do enforce, but it can't only be about enforcement—you need to decide within yourself to hold the line on integrity.*

Programming Abstractions in C++

WHAT ARE “PROGRAMMING
ABSTRACTIONS”?

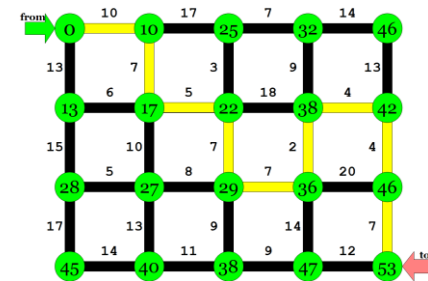
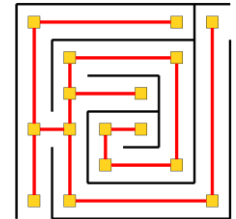
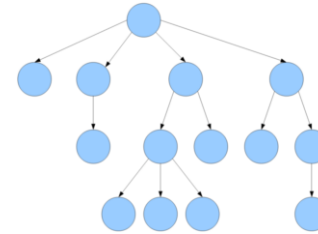
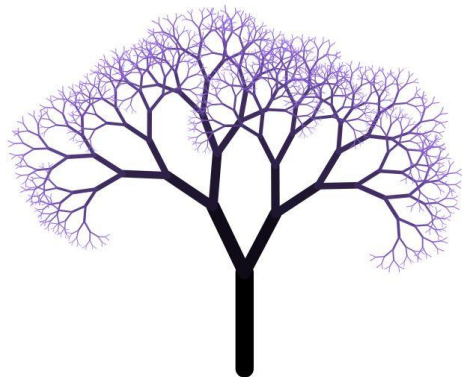


What is CS 106B?

CS 106B: Programming Abstractions

- solving big(ger) problems and processing big(ger) data
- learning to manage complex data structures
- algorithmic analysis and algorithmic techniques such as recursion
- programming style and software development practices
- familiarity with the C++ programming language

Prerequisite: CS 106A or equivalent



<http://cs106b.stanford.edu/>



Stanford University

What is this class
about?

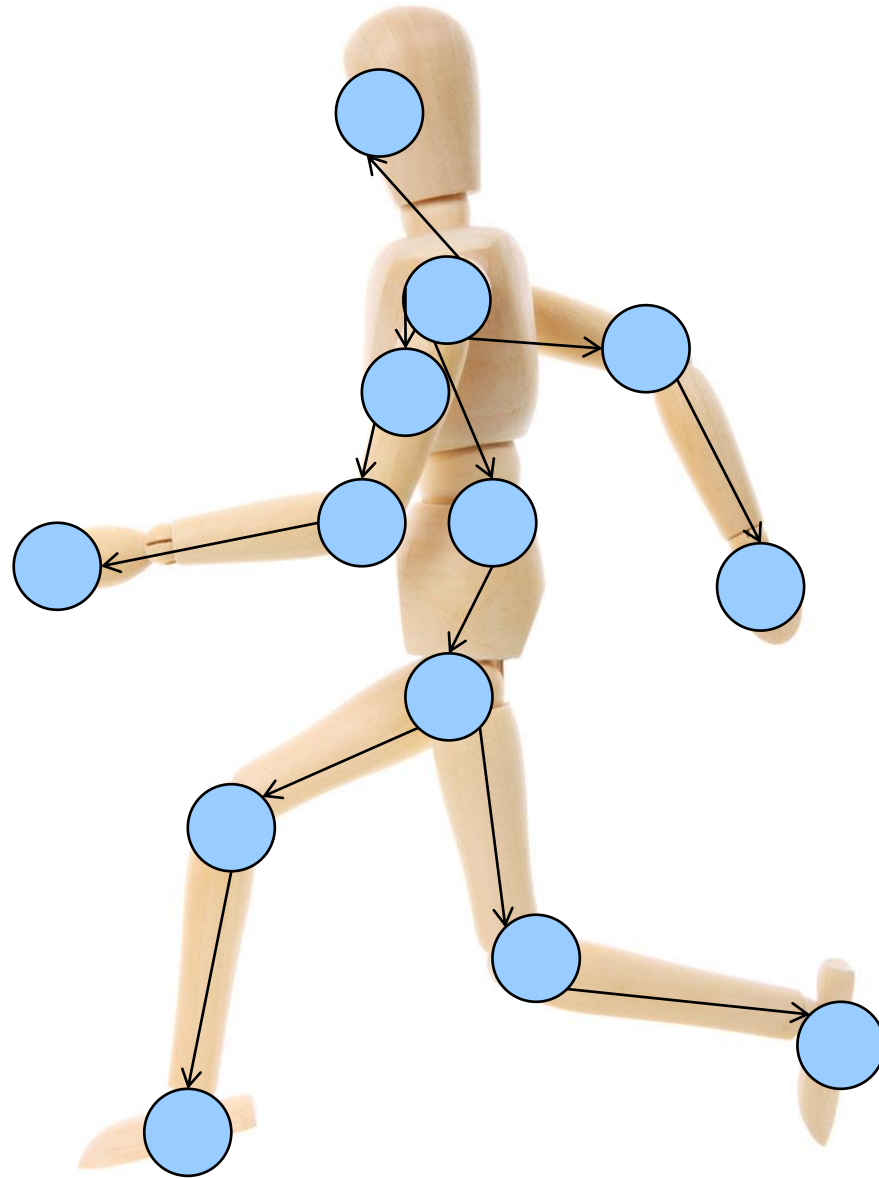
What do we mean by
“abstractions”?

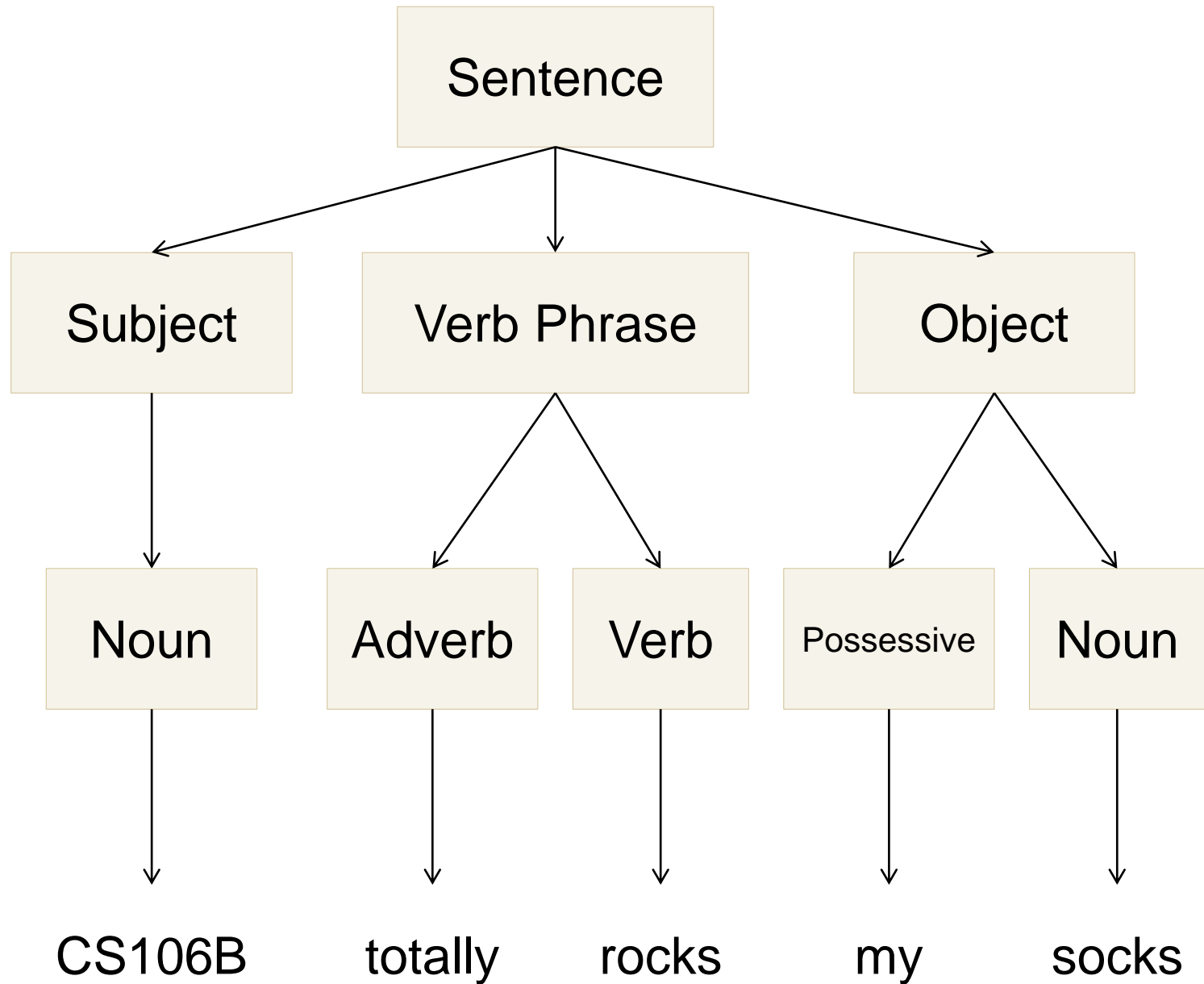


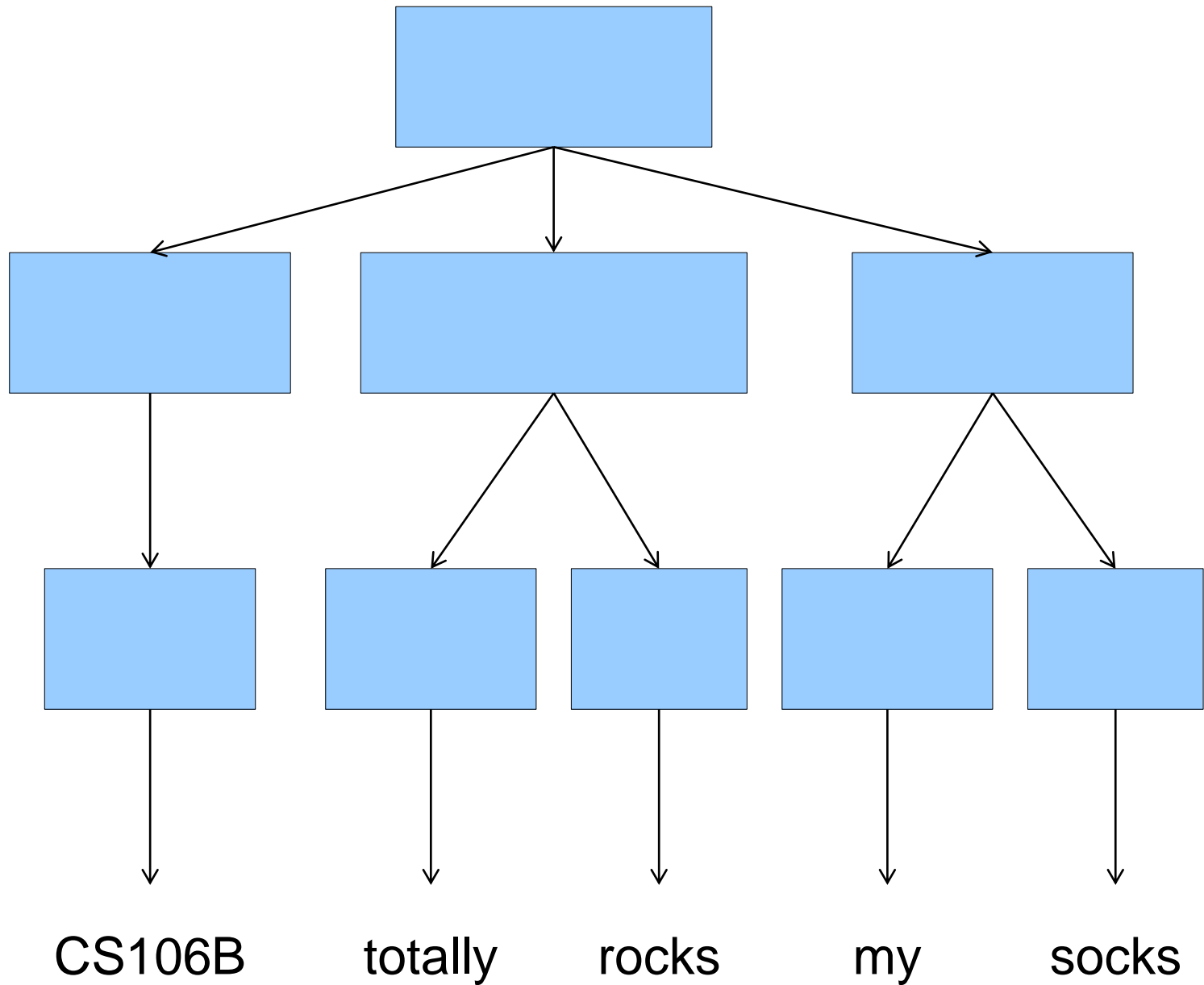
[Colatina](#), Carlos Nemer

This file is licensed under the [Creative Commons Attribution 3.0 Unported](#) license.

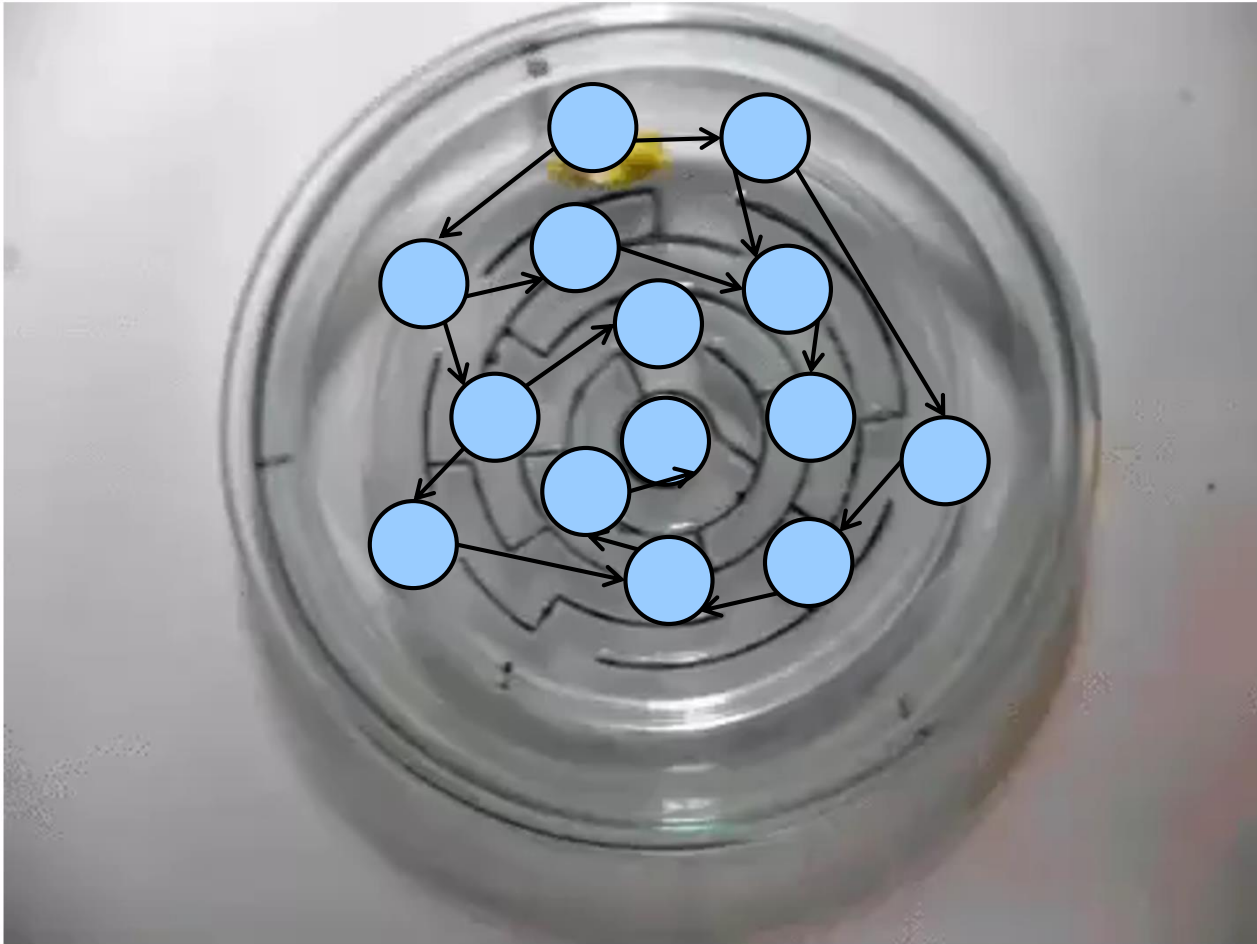


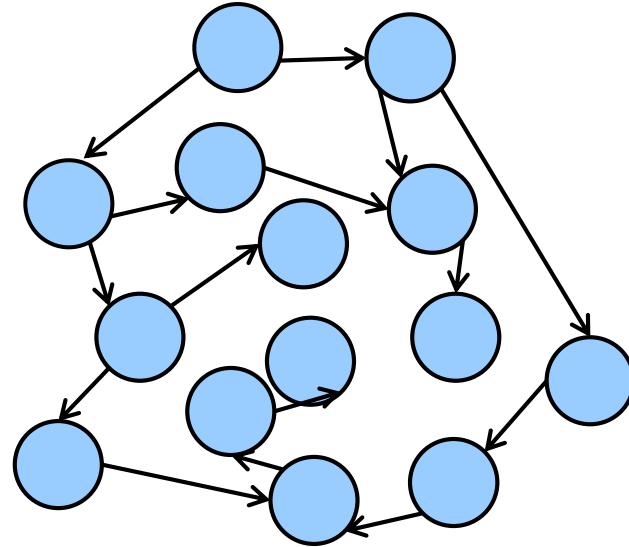






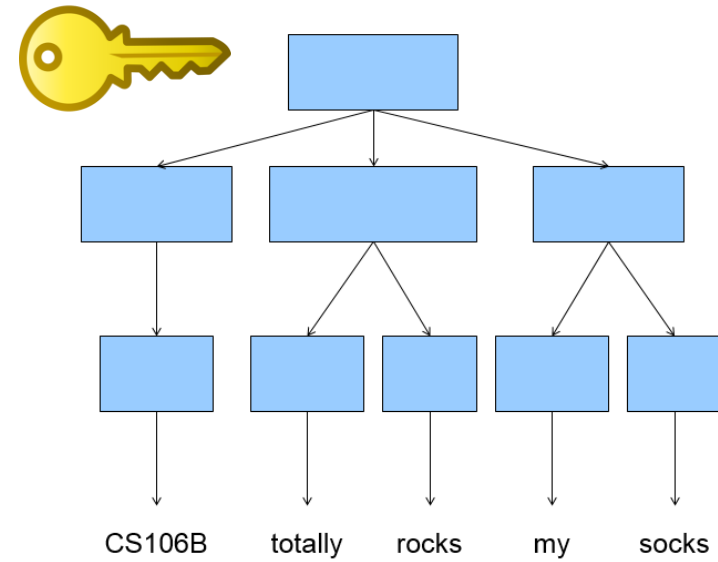
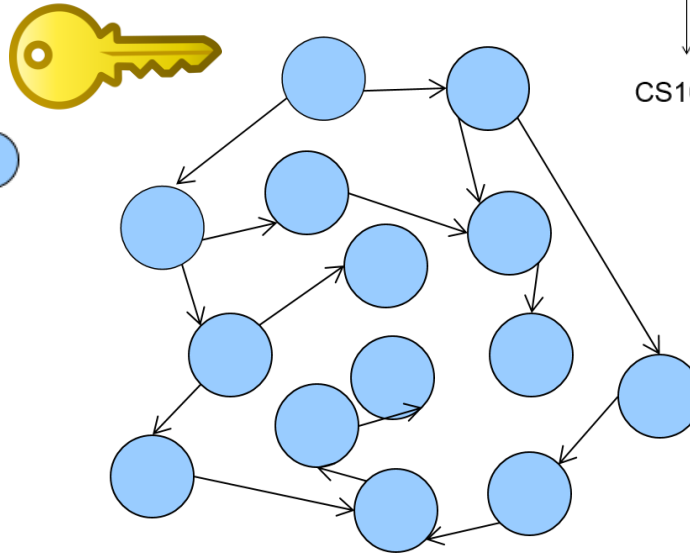
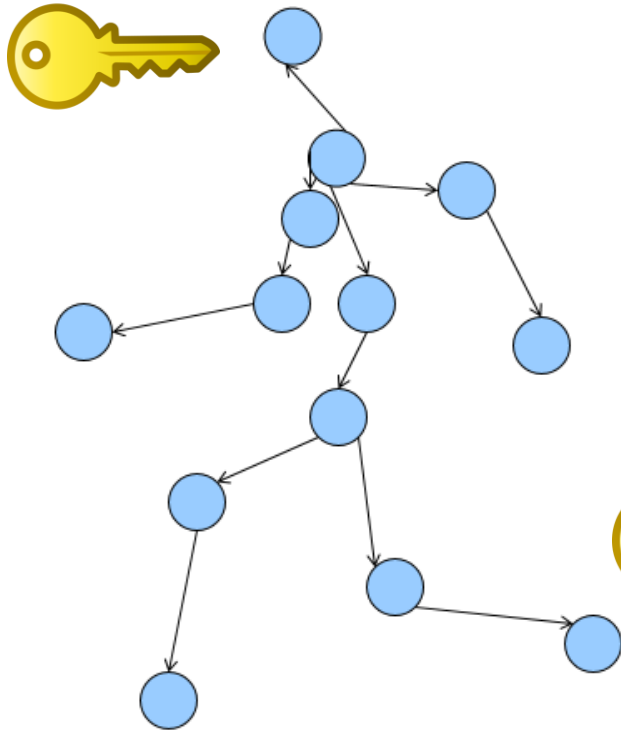






In CS106B, you'll learn to:

1. Identify common underlying structures
2. Apply known algorithmic tools that solve diverse problems that share that structure



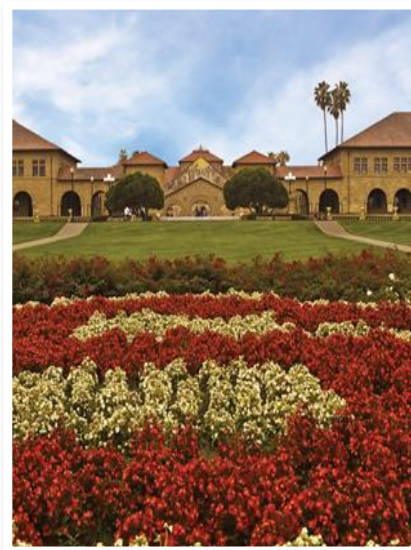
Building a vocabulary of **abstractions**
makes it possible to represent and solve a huge
variety of problems using known tools.

**After this course...
you'll have what is
effectively a superpower.**

**Spend some time thinking about how
you'll use it.**

Welcome to C++

LET'S START CODING!!



First C++ program (1.1)

```
/*  
 * hello.cpp  
 * This program prints a welcome message  
 * to the user.  
 */  
#include <iostream>  
#include "console.h"  
using namespace std;  
  
int main() {  
    cout << "Hello, world!" << endl;  
    return 0;  
}
```

Include statements are like imports in Java/Python. More on this in a moment.


Every C++ program has a **main** function. The program starts at **main** and executes its statements in sequence.

At program end, **main** returns 0 to indicate successful completion. A non-zero return value is an error code, but we won't use this method of error reporting in this class so we will always return zero.

C++ variables and types (1.5-1.8)

- The C++ compiler is rather picky about *types* when it comes to variables.
- Types exist in languages like Python (see the two code examples at right), but you don't need to say much about them in the code. They just happen.
- The **first time** you introduce a variable in C++, you need to announce its type to the compiler (what kind of data it will hold).
 - › After that, just use the variable name (don't repeat the type).
 - › You won't be able to change the type of data later! C++ variables can only do one thing.

C++



```
int x = 42 + 7 * -5;
double pi = 3.14159;
char letter = 'Q';
bool done = true;

x = x - 3;
```

Python

```
x = 42 + 7 * -5
pi = 3.14159
letter = 'Q'
done = True

x = x - 3
```

More C++ syntax examples (1.5-1.8)

```
for (int i = 0; i < 10; i++) {           // for loops
    if (i % 2 == 0) {                   // if statements
        x += i;
    }
}                                         /* two comment styles */
```

```
while (letter != 'Q' && !done) {        // while loops, logic
    x = x / 2;
    if (x == 42) { return 0; }
}
```

```
binky(pi, 17);                          // function call
winky("this is a string");               // string usage
```

Some C++ logistical details (2.2)

```
#include <libraryname>    // standard C++ library  
#include "libraryname.h" // local project library
```

- Attaches a library for use in your program
- Note the differences (common bugs):
 - <> vs " "
 - .h vs no .h

```
using namespace name;
```

- *Mostly, just don't worry about what this actually does/means! Copy & paste the std line below into the top of your programs.*
- Brings a group of features into global scope so your program can directly refer to them
- Many C++ standard library features are in namespace std so we write:
 - › using namespace std;
 - › “std” is short for “standard”