Welcome to CS106B!

Who's Here Today?

- Aeronautics and Astronautics
- Bioengineering
- Biology
- Chemical Engineering
- Chemistry
- Civil / Environmental Engineering
- Computational and Mathematical Engineering
- Computer Science
- Creative Writing
- Design
- Earth Planetary Sciences
- Earth Systems
- Economics
- Education
- Electrical Engineering
- Engineering
- Ethics in Society
- History
- Human Biology
- Immunology
- Law
- Mechanical Engineering
- Medicine
- Management Science and Entineering
- Physics
- Political Science
- Spanish
- Statistics
- Stem Cell Engineering / Regenerative Medicine
- Symbolic Systems
- Theater and Performance Studies
- ***Undeclared!***

Course Staff

Keith Schwarz
htiek@cs.stanford.edu

Jonathan Coronado
jonathan.coronado@stanford.edu

The CS106B Section Leaders

Prerequisites

CS106A

(or equivalent)
*(check out our **course placement page** if you're unsure!)*

Course Website

https://cs106b.stanford.edu

We also have a course Canvas site, which is mostly there for lecture videos and to link you to other resources.

Live Q&A

- Visit our EdStem page. It's linked through the course Canvas and also available here:

  https://edstem.org/us/courses/70781/discussion

- Next, find the pinned thread at the top entitled

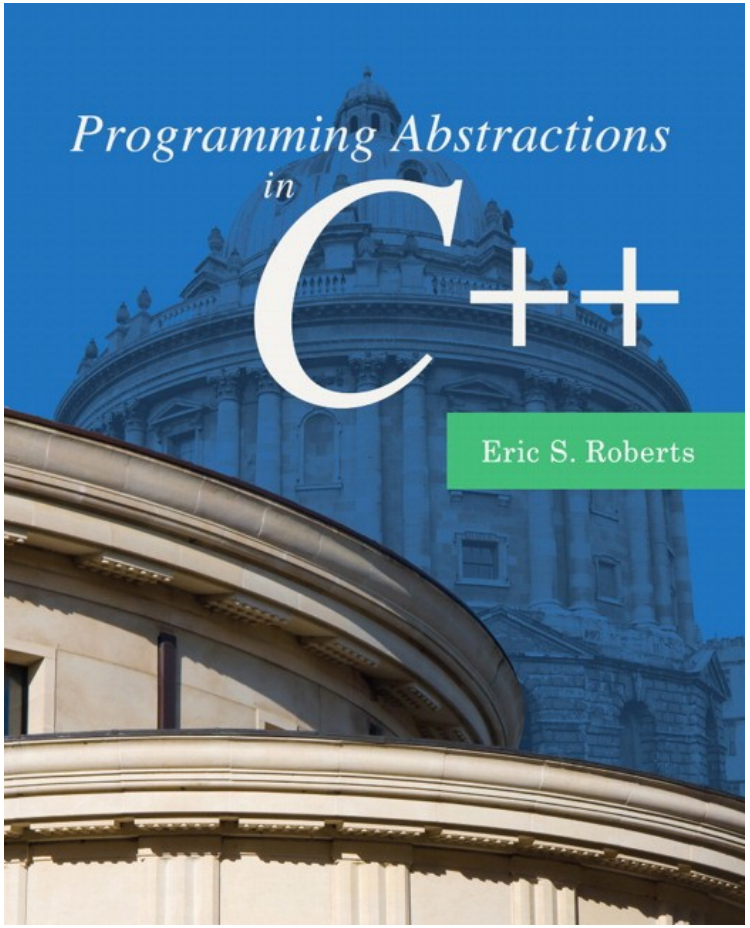
  L00: Introduction

- Once you've found that thread, give it a to let us know you've found it.
- Feel free to post questions here during lecture – we can then answer asynchronously.
- You're always welcome to raise your hand if you have any questions!

60-Minute Lectures

- We have an 80-minute time slot for lectures this quarter, but we'll only use 60 of those minutes (1:30PM – 2:30PM Pacific).
- Compared with a traditional 50-minute lecture, those extra ten minutes give us time to
  - answer your questions,
  - explore and tinker with code,
  - go at a more leisurely pace, and
  - let you play around with the material.
- I'll stick around for the remaining 20 minutes of our time block to chat with folks one-on-one about whatever it is that you're interested in.

Our Textbook



- Our course textbook is ***Programming Abstractions in C++*** by the legendary Eric Roberts.
  - There's a ***draft version*** available online.
- We've assigned readings for each lecture. You can either do them before or after the lectures – your choice.
- It's important to complete the readings in addition to attending lecture; there's a lot of really good info in there.

Discussion Sections

- Starting next week, we'll be holding weekly discussion sections.
- We have our own section signup system that is independent of the one run by Axess.
- Sign up between Thursday, January 9th at 5:00PM Pacific and Sunday, January 12th at 5:00PM Pacific by visiting

  https://cs198.stanford.edu/cs198/auth/default.aspx
- Looking forward: some of the later assignments can be done in pairs. ***You must be in the same section as someone to partner with them***. You may want to start thinking about folks you'd like to partner with.

Optional Add-Ons

- There are three one-unit courses you can optionally add on to CS106B.
- These are *in addition to* rather than *in place of* a regular discussion section.
  - CS100BACE offers additional practice and support with the material from CS106B in a small group setting. The application is ***available online here***.
  - CS106L provides a deep dive into the C++ programming language beyond what we'll cover in CS106B.
  - CS106S explores applications of the CS106B material to social good.
- Feel free to chat with us about these courses after class if you want to learn more!

Grading Policies

Grading Policies

40% Assignments

Eight Coding Assignments

Plus an intro assignment that goes out today and is due Friday.

Grading Policies

40% Assignments
20% Midterm Exam

Midterm Exam

Monday, February 10th
7PM – 10PM

Grading Policies

Final Exam

Monday, March 17th
8:30AM – 11:30AM

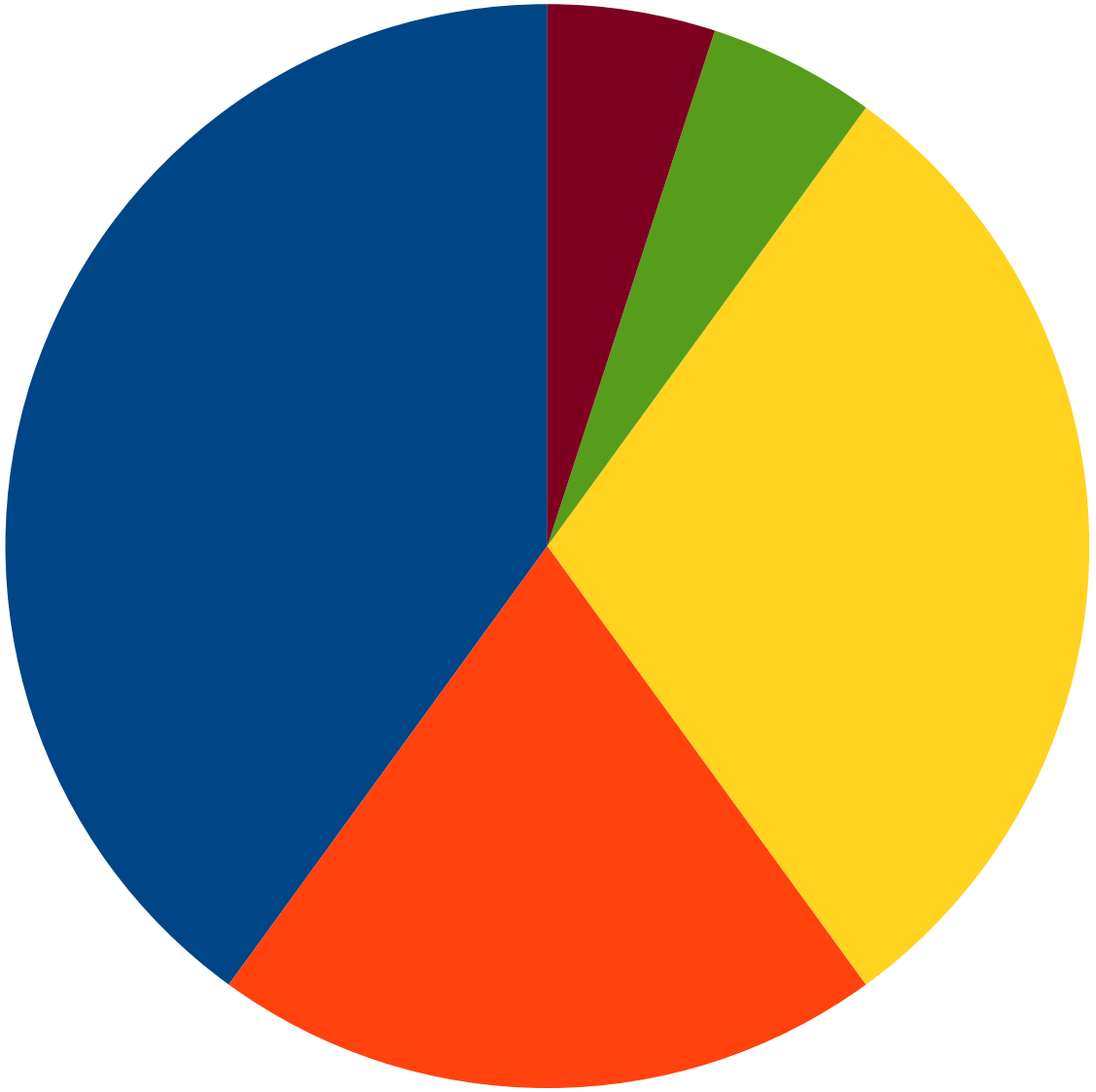
Grading Policies

- 40% Assignments
- 20% Midterm Exam
- 30% Final Exam
- 5% Section Participation

Discussion Sections

Our world-famous discussion sections!

Grading Policies



- 40% Assignments
- 20% Midterm Exam
- 30% Final Exam
- 5% Section Participation
- 5% Lecture Participation

Lecture Participation

Starts next week. We'll discuss details later this week.

What's Next in Computer Science?

Goals for this Course

- ***Learn how to model and solve complex problems with computers.***
- To that end:
  - Explore common abstractions for representing problems.
  - Harness recursion and understand how to think about problems recursively.
  - Quantitatively analyze different approaches for solving problems.

Goals for this Course

Learn how to model and solve complex problems with computers.

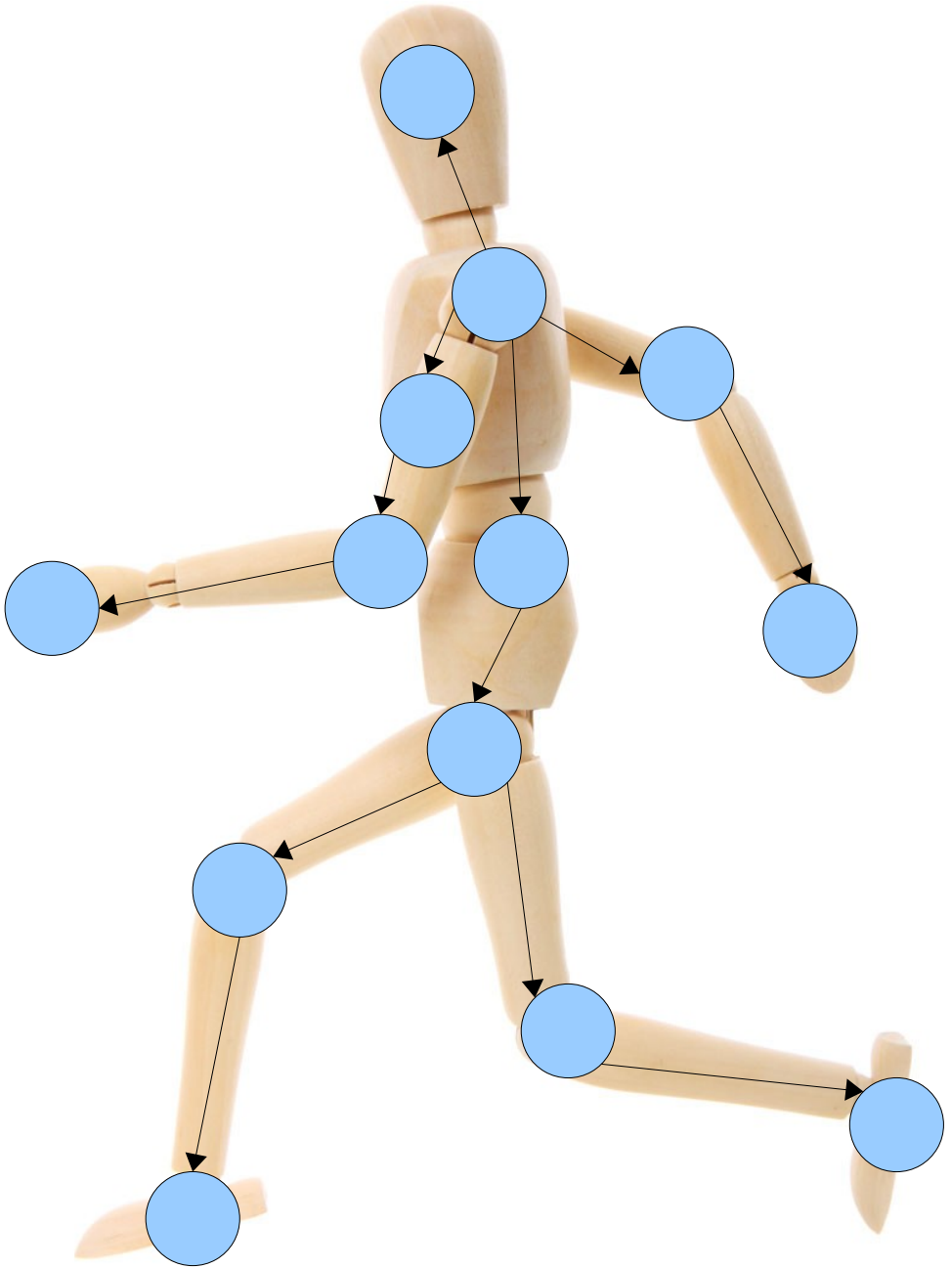
To that end:

- Explore common abstractions for representing problems.

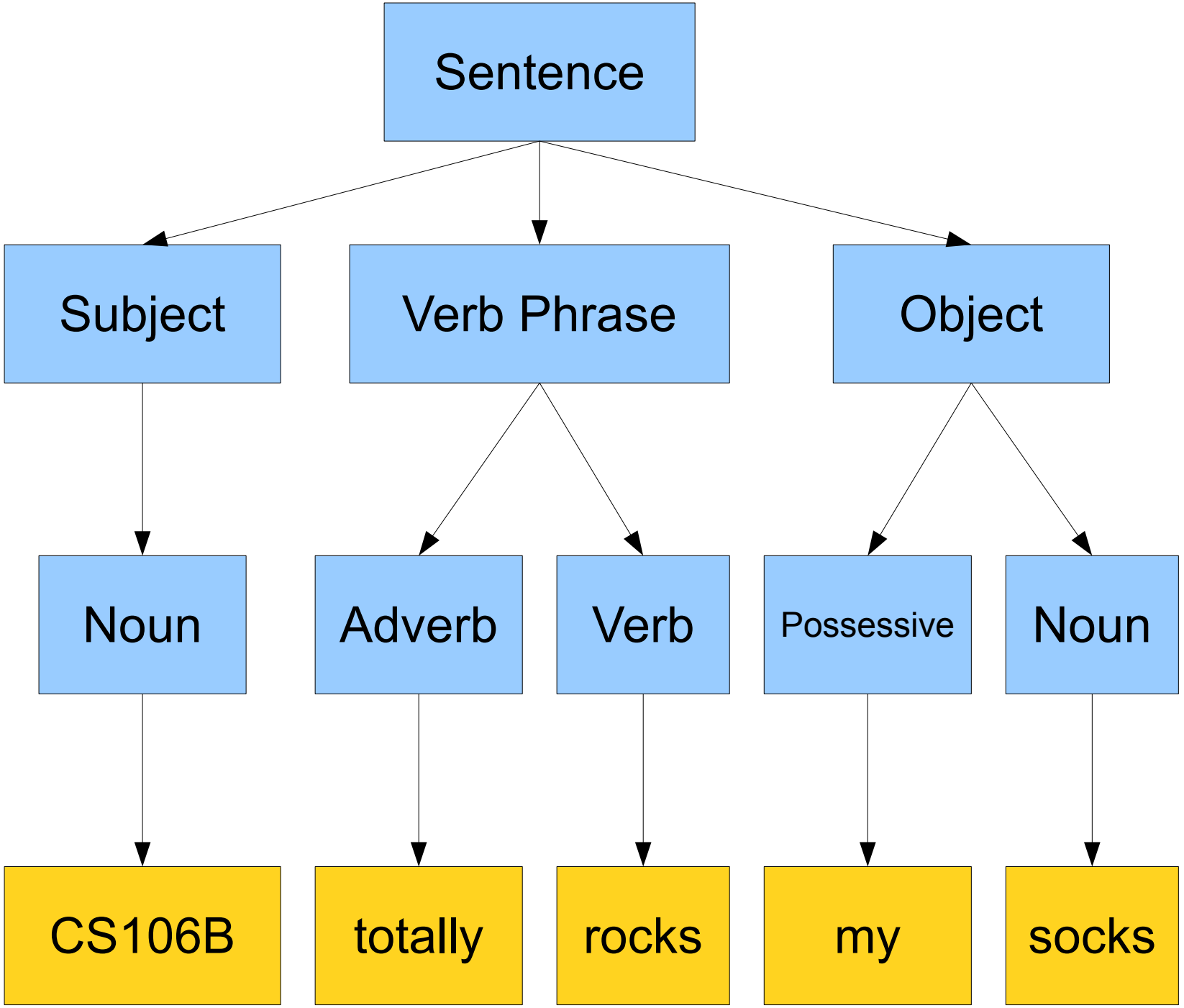
  Harness recursion and understand how to think about problems recursively.

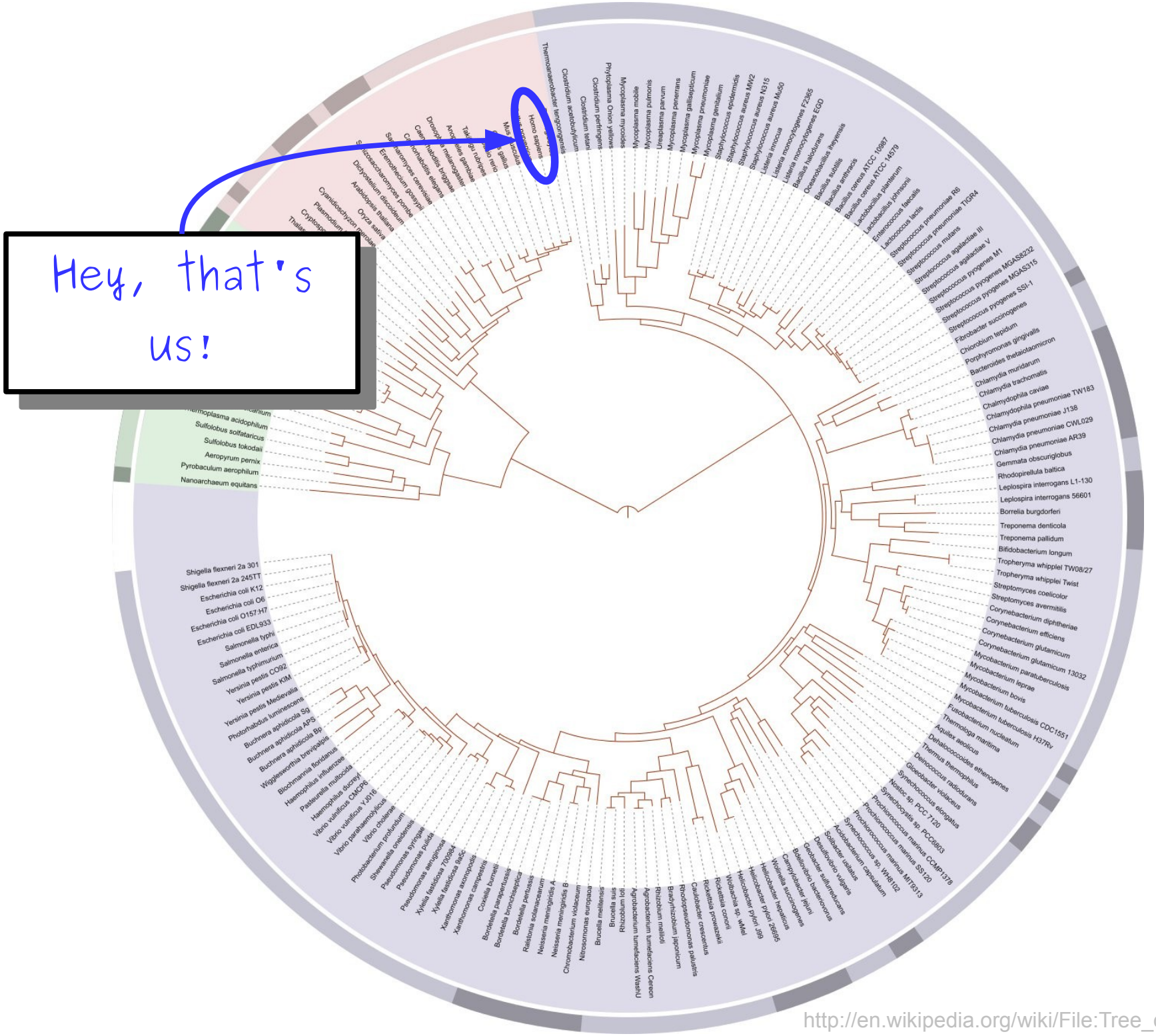
  Quantitatively analyze different approaches for solving problems.

http://www.publicdomainpictures.net/pictures/10000/velka/1-1265899974oKJ9.jpg

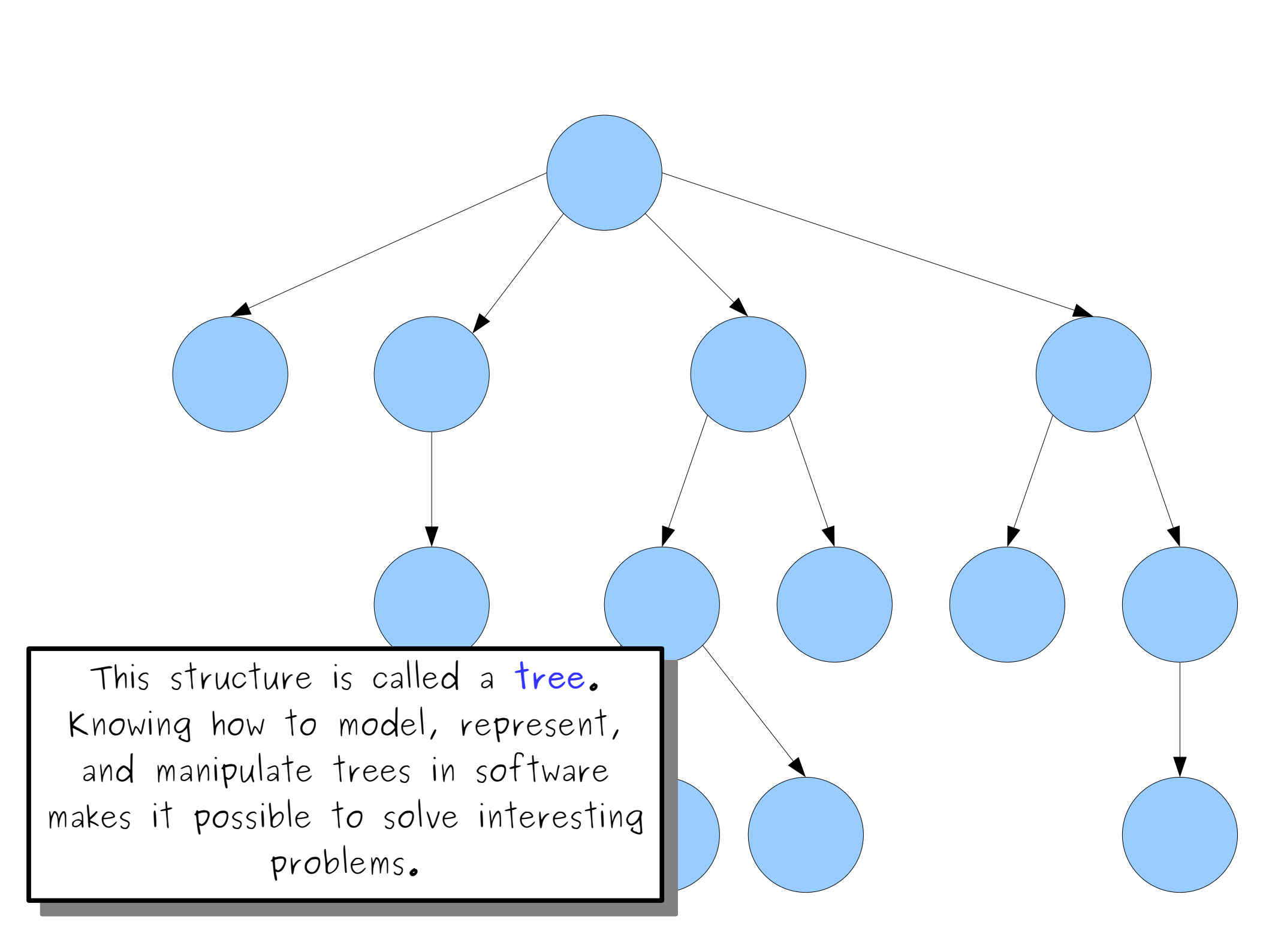


http://www.publicdomainpictures.net/pictures/10000/velka/1-1265899974oKJ9.jpg


Sentence
Subject
Verb Phrase
Object
Noun
Adverb
Verb
Possessive
Noun
CS106B
totally
rocks
my
socks



http://en.wikipedia.org/wiki/File:Tree_of_life_SVG.svg


This structure is called a tree.
Knowing how to model, represent,
and manipulate trees in software
makes it possible to solve interesting
problems.

Building a vocabulary of ***abstractions*** makes it possible to represent and solve a wider class of problems.

Goals for this Course

- ***Learn how to model and solve complex problems with computers.***
- To that end:
  - Explore common abstractions for representing problems.
  - Harness recursion and understand how to think about problems recursively.
  - Quantitatively analyze different approaches for solving problems.

Goals for this Course

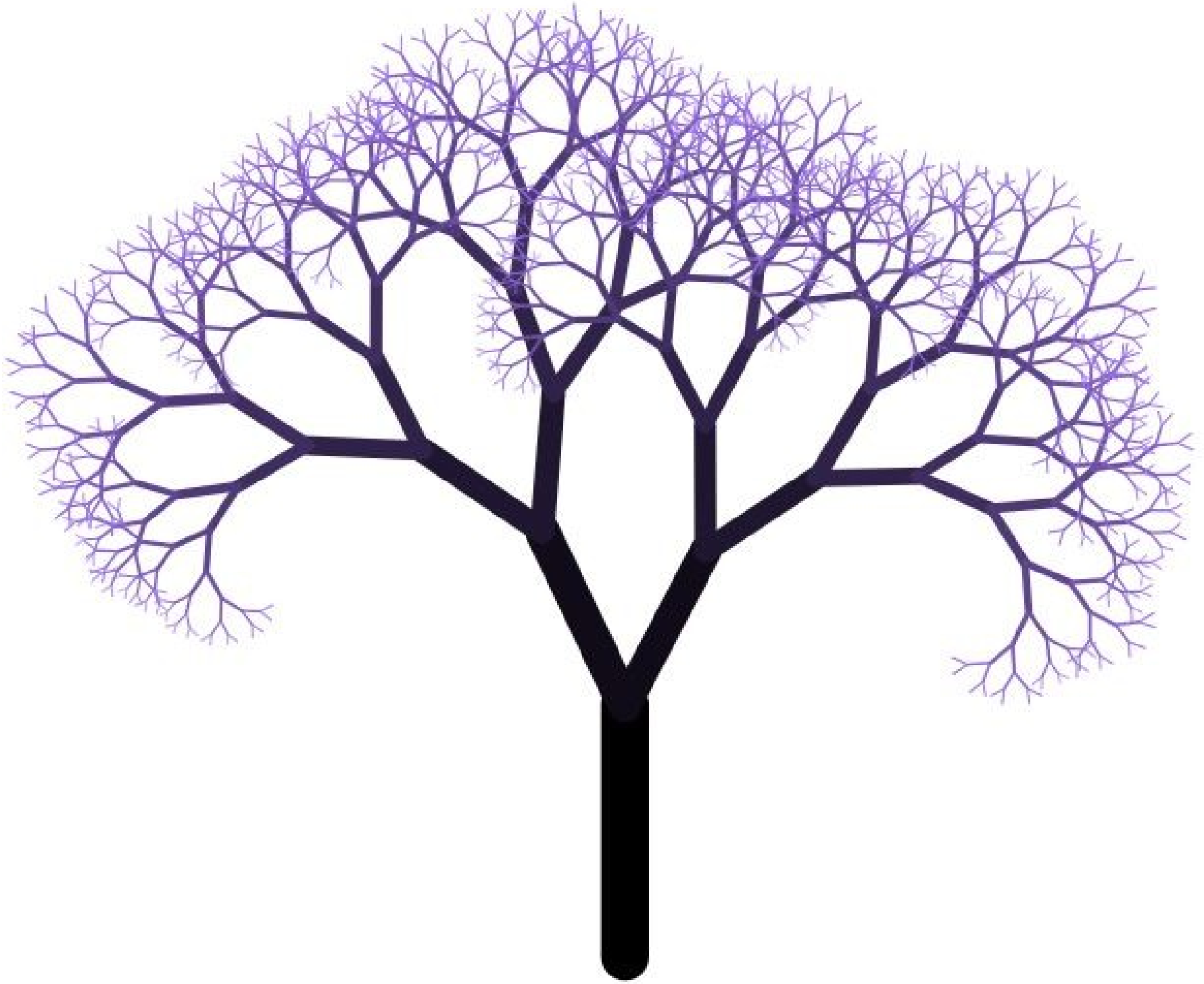
Learn how to model and solve complex problems with computers.

To that end:

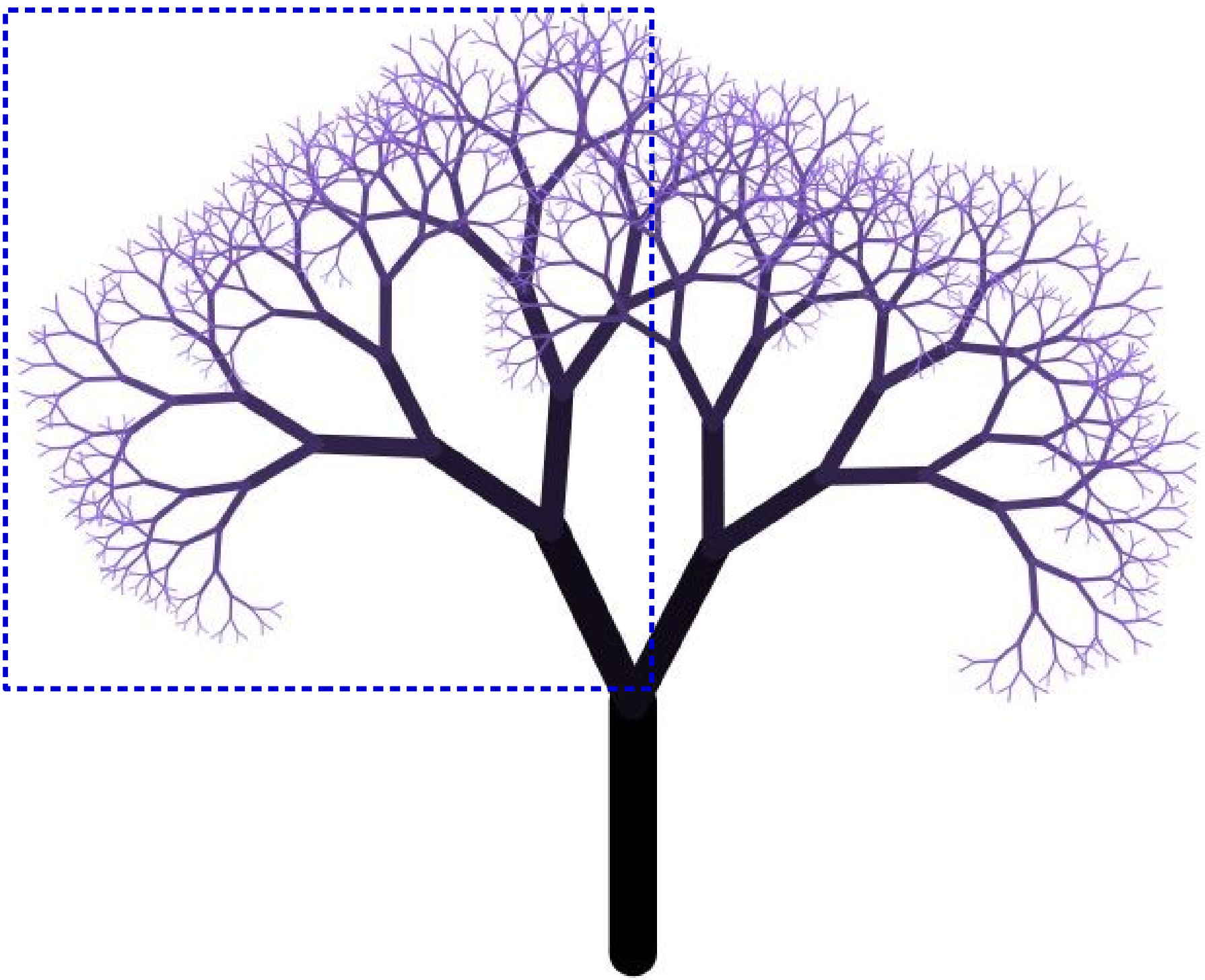
Explore common abstractions for representing problems.

- Harness recursion and understand how to think about problems recursively.

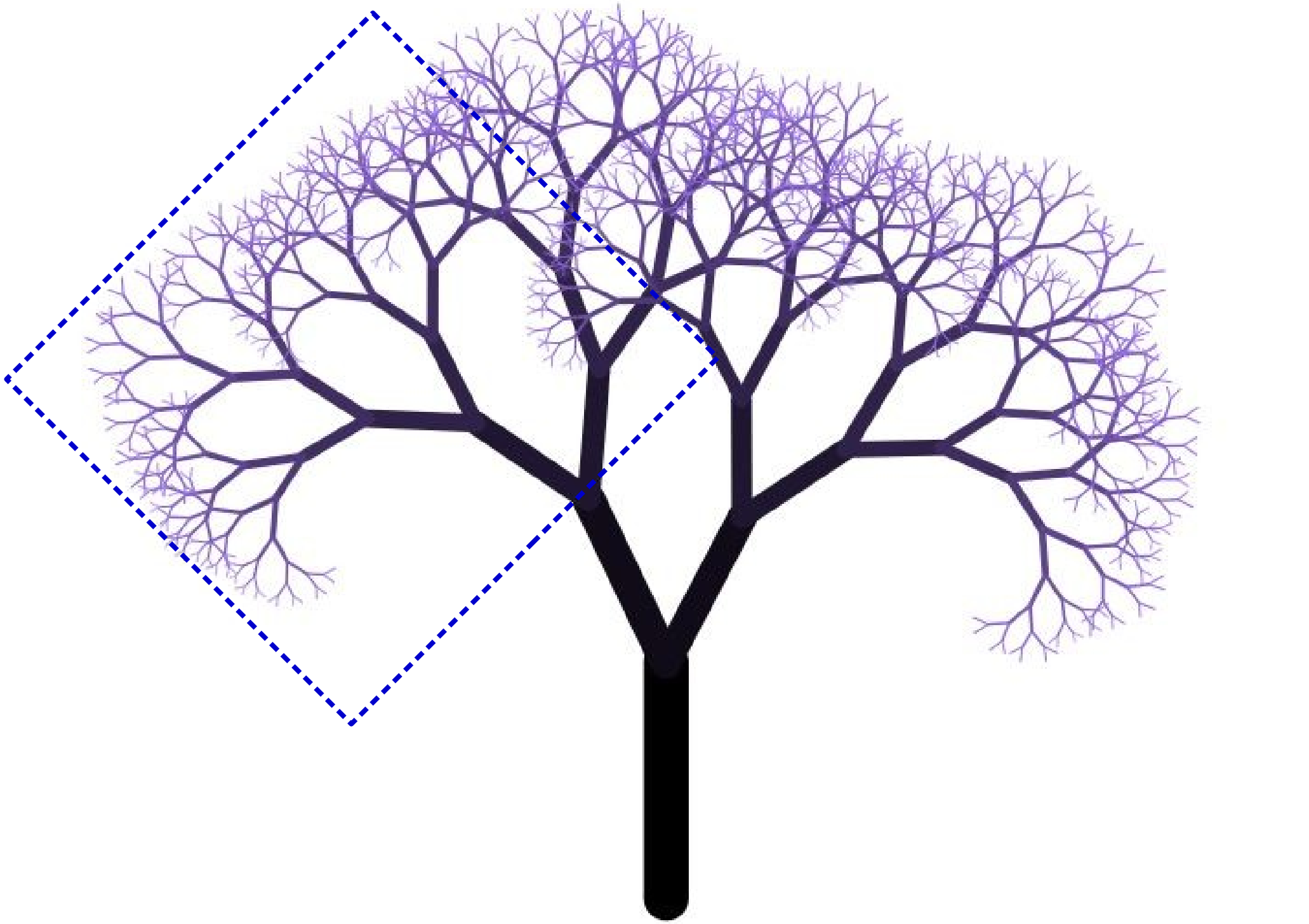
Quantitatively analyze different approaches for solving problems.



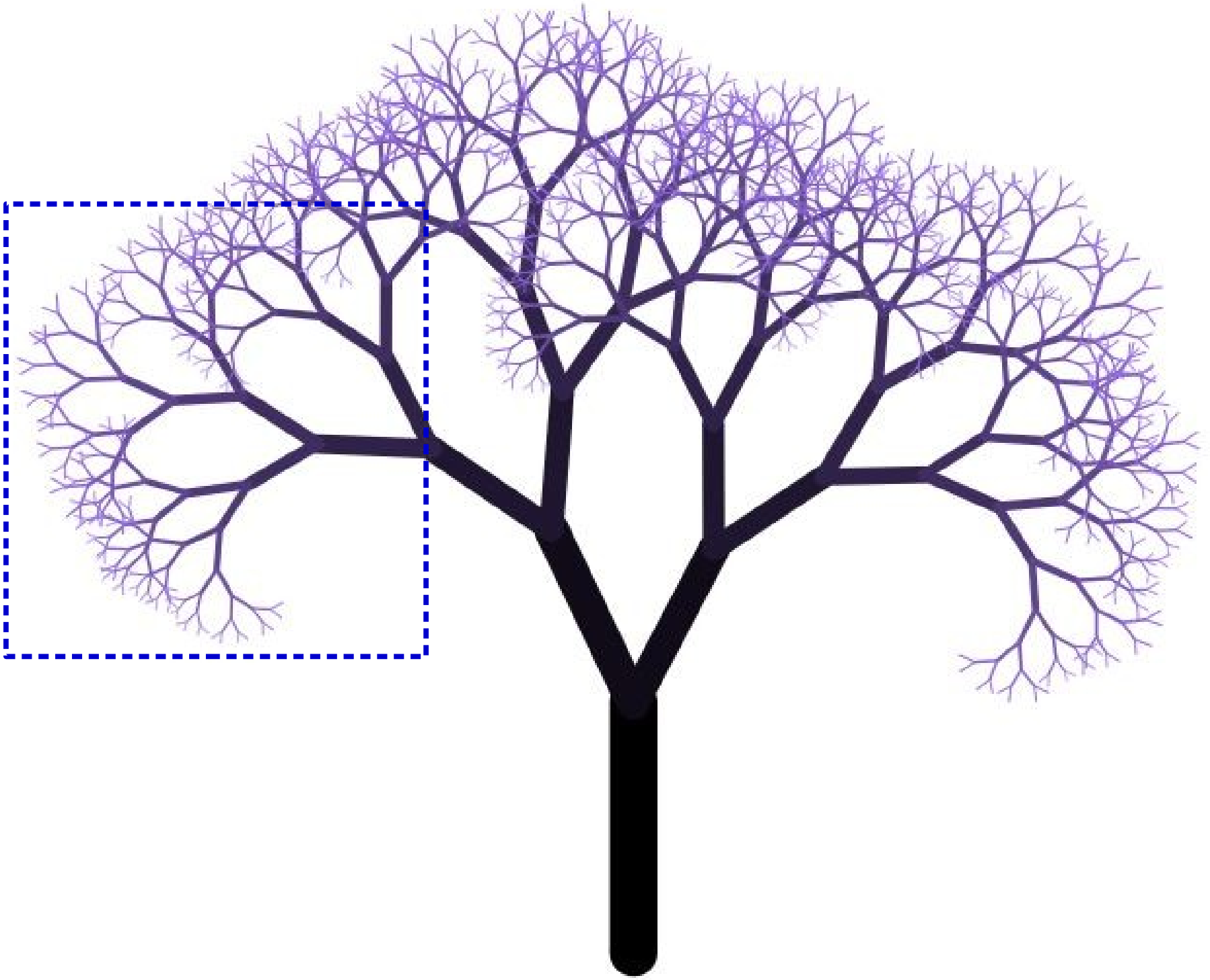
http://www.marketoracle.co.uk/images/2010/Oct/fractal-tree2.jpg



http://www.marketoracle.co.uk/images/2010/Oct/fractal-tree2.jpg



http://www.marketoracle.co.uk/images/2010/Oct/fractal-tree2.jpg



http://www.marketoracle.co.uk/images/2010/Oct/fractal-tree2.jpg

A ***recursive solution*** is a solution that is defined in terms of itself.

Goals for this Course

- ***Learn how to model and solve complex problems with computers.***
- To that end:
  - Explore common abstractions for representing problems.
  - Harness recursion and understand how to think about problems recursively.
  - Quantitatively analyze different approaches for solving problems.

Goals for this Course

Learn how to model and solve complex problems with computers.

To that end:

Explore common abstractions for representing problems.

Harness recursion and understand how to think about problems recursively.

- Quantitatively analyze different approaches for solving problems.

ull,"status":"reviewed","tsunami":0,"sig":369,"net":"us","code":"2000j048","ids":",us2000j
origin,phase-data,","nst":null,"dmin":1.598,"rms":0.78,"gap":104,"magType":"mww","type":"e
Tobelo, Indonesia"},"geometry":{"type":"Point","coordinates":[127.3157,2.3801,53.72]},"id"
{"type":"Feature","properties":{"mag":5.1,"place":"265km SW of Severo-Kuril'sk,
Russia","time":1546548377590,"updated":1546549398040,"tz":600,"url":"https://earthquake.us
detail":"https://earthquake.usgs.gov/earthquakes/feed/v1.0/detail/us2000j03t.geojson","fel
,"status":"reviewed","tsunami":0,"sig":400,"net":"us","code":"2000j03t","ids":",us2000j03t
gin,phase-data,","nst":null,"dmin":5.198,"rms":0.94,"gap":48,"magType":"mww","type":"earth
Severo-Kuril'sk, Russia"},"geometry":{"type":"Point","coordinates":[153.7105,48.8712,104.7
{"type":"Feature","properties":{"mag":4.8,"place":"20km NNW of Taitung City,
Taiwan","time":1546538570070,"updated":1546541624040,"tz":480,"url":"https://earthquake.us
detail":"https://earthquake.usgs.gov/earthquakes/feed/v1.0/detail/us2000j02k.geojson","fel
,"status":"reviewed","tsunami":0,"sig":354,"net":"us","code":"2000j02k","ids":",us2000j02k
gin,phase-data,","nst":null,"dmin":0.52,"rms":0.79,"gap":110,"magType":"mb","type":"earthq
City, Taiwan"},"geometry":{"type":"Point","coordinates":[121.0489,22.9222,10]},"id":"us200
{"type":"Feature","properties":{"mag":5,"place":"79km ENE of Petropavlovsk-Kamchatskiy,
Russia","time":1546538266300,"updated":1546541474965,"tz":720,"url":"https://earthquake.us
detail":"https://earthquake.usgs.gov/earthquakes/feed/v1.0/detail/us2000j02g.geojson","fel
us":"reviewed","tsunami":0,"sig":385,"net":"us","code":"2000j02g","ids":",us2000j02g,","so
in,phase-data,","nst":null,"dmin":0.728,"rms":0.75,"gap":114,"magType":"mb","type":"earthq
Petropavlovsk-Kamchatskiy, Russia"},"geometry":{"type":"Point","coordinates":[159.6844,53.
{"type":"Feature","properties":{"mag":4.5,"place":"South of Java,
Indonesia","time":1546533739000,"updated":1546539809085,"tz":420,"url":"https://earthquake
","detail":"https://earthquake.usgs.gov/earthquakes/feed/v1.0/detail/us2000j024.geojson","
tatus":"reviewed","tsunami":0,"sig":312,"net":"us","code":"2000j024","ids":",us2000j024,",
rigin,phase-data,","nst":null,"dmin":2.821,"rms":0.89,"gap":83,"magType":"mb","type":"eart
Indonesia"},"geometry":{"type":"Point","coordinates":[108.5165,-10.6419,8.84]},"id":"us200
{"type":"Feature","properties":{"mag":4.8,"place":"108km N of Ishigaki,
Japan","time":1546529675300,"updated":1546530815040,"tz":480,"url":"https://earthquake.usg
etail":"https://earthquake.usgs.gov/earthquakes/feed/v1.0/detail/us2000j01x.geojson","felt
"status":"reviewed","tsunami":0,"sig":354,"net":"us","code":"2000j01x","ids":",us2000j01x,
in,phase-data,","nst":null,"dmin":1.342,"rms":0.82,"gap":68,"magType":"mb","type":"earthqu
Japan"},"geometry":{"type":"Point","coordinates":[124.1559,25.3209,122.33]},"id":"us2000j0
{"type":"Feature","properties":{"mag":5.4,"place":"82km S of Bristol Island, South Sandwic
Islands","time":1546519662810,"updated":1546520523040,"tz":-120,"url":"https://earthquake.
,"detail":"https://earthquake.usgs.gov/earthquakes/feed/v1.0/detail/us2000j01b.geojson","f

There are many ways to solve the same problem. How do we ***quantitatively*** talk about how they compare?

Goals for this Course

- ***Learn how to model and solve complex problems with computers.***
- To that end:
  - Explore common abstractions for representing problems.
  - Harness recursion and understand how to think about problems recursively.
  - Quantitatively analyze different approaches for solving problems.

Who's Here Today?

- Aeronautics and Astronautics
- Bioengineering
- Biology
- Chemical Engineering
- Chemistry
- Civil / Environmental Engineering
- Computational and Mathematical Engineering
- Computer Science
- Creative Writing
- Design
- Earth Planetary Sciences
- Earth Systems
- Economics
- Education
- Electrical Engineering
- Engineering
- Ethics in Society
- History
- Human Biology
- Immunology
- Law
- Mechanical Engineering
- Medicine
- Management Science and Entineering
- Physics
- Political Science
- Spanish
- Statistics
- Stem Cell Engineering / Regenerative Medicine
- Symbolic Systems
- Theater and Performance Studies
- ***Undeclared!***

Transitioning to C++

Transitioning to C++

- I’m assuming that the majority of you are either coming out of CS106A in Python coming from AP CS in Java.
- In this course, we’ll use the C++ programming language.
- Learning a second programming language is ***substantially*** easier than learning a first.
  - You already know how to solve problems; you just need to adjust the syntax you use.
- While the languages are superficially different, they have much in common.

Our First C++ Program

Prime Numbers

- A positive integer n is called a ***prime number*** if it's greater than one and its only positive divisors are 1 and n.
- For example:
  - 15 isn't prime ($15 = 3 \times 5$).
  - 17 is prime.
  - 19 is prime.
  - 21 isn't prime ($21 = 3 \times 7$).
- What's the $5{,}000^{\text{th}}$ prime number?

```
def isPrime(n):
    """Returns whether the input number is prime; assumes n >= 2."""

    # Try dividing by all numbers from 2 to n - 1, inclusive.
    for divisor in range(2, n):
        if n % divisor == 0:
            return False



    return True



found = 0   # How many prime numbers we've found
number = 1  # Next number to test

# Keep trying numbers until we've found 5000 primes.
while found < 5000:
    number += 1

    if isPrime(number):
        found += 1



# Last number tried is the 5000th prime.
print(number)
```

```cpp
#include <iostream>
using namespace std;

/* Returns whether the input number is prime; assumes n >= 2. */
bool isPrime(int n) {
    /* Try dividing by all numbers from 2 to n - 1, inclusive. */
    for (int divisor = 2; divisor < n; divisor++) {
        if (n % divisor == 0) {
            return false;
        }
    }

    return true;
}

int main() {
    int found = 0;  // How many prime numbers we've found
    int number = 1; // Next number to test

    /* Keep trying numbers until we've found 5000 primes. */
    while (found < 5000) {
        number++;

        if (isPrime(number)) {
            found++;
        }
    }

    /* Last number tried is the 5000th prime. */
    cout << number << endl;
    return 0;
}
```

```cpp
#include <iostream>
using namespace std;

/* Returns whether the input number is prime; assumes n >= 2. */
bool isPrime(int n) {
    /* Try dividing by all numbers from 2 to n - 1, inclusive. */
    for (int divisor = 2; divisor < n; divisor++) {
        if (n % divisor == 0) {
            return false;
        }
    }

    return true;
}

int main() {
    int found = 0;  // How many prime numbers we've found
    int number = 1; // Next number to test

    /* Keep trying numbers until we've found 5000 primes. */
    while (found < 5000) {
        number++;

        if (isPrime(number)) {
            found++;
        }
    }

    /* Last number tried is the 5000th prime. */
    cout << number << endl;
    return 0;
}
```

In Python, indentation alone determines nesting.

In C++, indentation is nice, but ***curly braces*** alone determine nesting.

```cpp
#include <iostream>
using namespace std;

/* Returns whether the input number is prime; assumes n >= 2. */
bool isPrime(int n) {
    /* Try dividing by all numbers from 2 to n - 1, inclusive. */
    for (int divisor = 2; divisor < n; divisor++) {
        if (n % divisor == 0) {
            return false;
        }
    }

    return true;
}

int main() {
    int found = 0;  // How many prime numbers we've found
    int number = 1; // Next number to test

    /* Keep trying numbers until we've found 5000 primes. */
    while (found < 5000) {
        number++;

        if (isPrime(number)) {
            found++;
        }
    }

    /* Last number tried is the 5000th prime. */
    cout << number << endl;
    return 0;
}
```

Python uses **True** and **False**;
C++ uses **true** and **false**.

```
#include <iostream>
using namespace std;

/* Returns whether the input number is prime; assumes n >= 2. */
bool isPrime(int n) {
    /* Try dividing by all numbers from 2 to n
    for (int divisor = 2; divisor < n; divisor
        if (n % divisor == 0) {
            return false;
        }
    }

    return true;
}

int main() {
    int found = 0;  // H
    int number = 1; // N

    /* Keep trying numbe
    while (found < 5000)
        number++;

        if (isPrime(number)) {
            found++;
        }
    }

    /* Last number tried is the 5000th prime. */
    cout << number << endl;
    return 0;
}
```

In Python, newlines mark the end of statements.

In C++, individual statements must have a semicolon (;) after them.

```
return false;
```

```cpp
#include <iostream>
using namespace std;

/* Returns whether the input number is prime; assumes n >= 2. */
bool isPrime(int n) {
    /* Try dividing by all numbers from 2 to n - 1, inclusive. */
    for (int divisor = 2; divisor < n; divisor++) {
        if (n % divisor == 0) {
            return false;
        }
    }

    return true;
}

int main() {
    int found = 0;  // How many prime numbers we've found
    int number = 1; // Next number to test

    /* Keep trying numbers until we've found 5000 primes. */
    while (found < 5000) {
        number++;

        if (isPrime(number)) {
            found++;
        }
    }

    /* Last number tried is th
    cout << number << endl;
    return 0;
}
```

In Python, you print output by using `print()`.

In C++, you use the ***stream insertion operator*** (`<<`) to push data to the console. (Pushing `endl` prints a newline.)

```cpp
#include <iostream>
using namespace std;

/* Returns whether the input number is prime; assumes n >= 2. */
bool isPrime(int n) {
    /* Try dividing by all numbers from
    for (int divisor = 2; divisor < n; d
        if (n % divisor == 0) {
            return false;
        }
    }

    return true;
}

int main() {
    int found = 0;  // How many prime numbers we've found
    int number = 1; // Next number to test

    /* Keep trying numbers until we've found 5000 primes. */
    while (found < 5000) {
        number++;

        if (isPrime(number)) {
            found++;
        }
    }

    /* Last number tried is the 5000th prime. */
    cout << number << endl;
    return 0;
}
```

In Python, you can optionally put parentheses around conditions in `if` statements and `while` loops.

In C++, these are mandatory.

```cpp
#include <iostream>
using namespace std;

/* Returns whether the input number is prime; assumes n >= 2. */
bool isPrime(int n) {
    /* Try dividing by all numbers from 2 to n - 1, inclusive. */
    for (int divisor = 2; divisor < n; divisor++) {
        if (n % divisor == 0) {
            return false;
        }
    }

    return true;
}

int main() {
    int found = 0;  // How many prime numbers we've found
    int number = 1; // Next number to test

    /* Keep trying numbers until we've found 5000 primes. */
    while (found < 5000) {
        number++;

        if (isPrime(number)) {
            found++;
        }
    }

    /* Last number tried is the 5000th prime. */
    cout << number << endl;
    return 0;
}
```

Python and C++ each have `for` loops, but the syntax is different. (Check the textbook for more details about how this works!)

```
#include <iostream>
using namespace std;

/* Returns whether the input number is prime; assumes n >= 2. */
bool isPrime(int n) {
    /* Try dividing by all numbers from 2 to n - 1, inclusive. */
    for (int divisor = 2; divisor < n; divisor++) {
        if (n % divisor == 0) {
            return false;
        }
    }

    return true;
}

int main() {
    int found = 0;  // How many prim
    int number = 1; // Next number to test

    /* Keep trying numbers until we've found 5000 primes. */
    while (found < 5000) {
        number++;

        if (isPrime(number)) {
            found++;
        }
    }

    /* Last number tried is the 5000th prime. */
    cout << number << endl;
    return 0;
}
```

C++ has an operator ++ that means “change this variable’s value by adding one to it.” Python doesn’t have this.

```cpp
#include <iostream>
using namespace std;

/* Returns whether the input number is prime; assumes n >= 2. */
bool isPrime(int n) {
    /* Try dividing by all numbers from 2 to n - 1, inclusive. */
    for (int divisor = 2; divisor < n; divisor++) {
        if (n % divisor == 0) {
            return false;
        }
    }

    return true;
}

int main() {
    int found = 0;  // How many prime numbers we've found
    int number = 1; // Next number to test

    /* Keep trying numbers until we've found 5000 primes. */
    while (found < 5000) {
        number++;

        if (isPrime(number)) {
            found++;
        }
    }

    /* Last number tried is the 5000th prime. */
    cout << number << endl;
    return 0;
}
```

In Python, comments start with # and continue to the end of the line.

In C++, there are two styles of comments. Comments starting with /* continue until */. Comments starting with // continue to the end of the line.

```
#include <iostream>
using namespace std;

/* Returns whether the input number is prime; assumes n >= 2. */
bool isPrime(int n) {
    /* Try dividing by all numbers from 2 to n - 1, inclusive. */
    for (int divisor = 2; divisor < n;
        if (n % divisor == 0) {
            return false;
        }
    }

    return true;
}

int main() {
    int found = 0;  // How many prime numbers we've found
    int number = 1; // Next number to test

    /* Keep trying numbers until we've found 5000 primes. */
    while (found < 5000) {
        number++;

        if (isPrime(number)) {
            found++;
        }
    }

    /* Last number tried is the 5000th prime. */
    cout << number << endl;
    return 0;
}
```

In Python, comments on functions are customarily written after the first line of the function.

In C++, comments on functions are customarily written *before* the first line of the function.

```cpp
#include <iostream>
using namespace std;

/* Returns whether the input number is prime; assumes n >= 2. */
bool isPrime(int n) {
    /* Try dividing by all numbers from 2 to n - 1, inclusive. */
    for (int divisor = 2; divisor < n; divisor
        if (n % divisor == 0) {
            return false;
        }
    }

    return true;
}

int main() {
    int found = 0;  // How many prime numbers
    int number = 1; // Next number to test

    /* Keep trying numbers until we've found 5000 primes. */
    while (found < 5000) {
        number++;

        if (isPrime(number)) {
            found++;
        }
    }

    /* Last number tried is the 5000th prime. */
    cout << number << endl;
    return 0;
}
```

In Python, each object has a type, but it isn't stated explicitly.

In C++, you *must* give a type to each variable. (The `int` type represents an integer, and `bool` represents a Boolean true or false.)

```
#include <iostream>
using namespace std;

/* Returns whether the input number is prime; assumes n >= 2. */
bool isPrime(int n) {
    /* Try dividing by all numbers fro
    for (int divisor = 2; divisor < n;
        if (n % divisor == 0) {
            return false;
        }
    }

    return true;
}

int main() {
    int found = 0;  // How many prime numbers we've found
    int number = 1; // Next number to test

    /* Keep trying numbers until we've found 5000 primes. */
    while (found < 5000) {
        number++;

        if (isPrime(number)) {
            found++;
        }
    }

    /* Last number tried is the 5000th prime. */
    cout << number << endl;
    return 0;
}
```

In Python, statements can be either in a function or at the top level of the program.

In C++, most statements must be inside of a function.

Why do we have both C++ and Python?

C++ and Python

- Python is a *great* language for data processing and writing quick scripts across all disciplines.
  - It's pretty quick to make changes to Python programs and then run them to see what's different.
  - Python programs, generally, run more slowly than C++ programs.
- C++ is a *great* language for writing high-performance code that takes advantage of underlying hardware.
  - Compiling C++ code introduces some delays between changing the code and running the code.
  - C++ programs, generally, run much faster than Python programs.
- Knowing both languages helps you use the right tool for the right job.

Your Action Items

- ***Read Chapter 1 of the textbook.***
  - Use this as an opportunity to get comfortable with the basics of C++ programming and to read more examples of C++ code.
- ***Start Assignment 0.***
  - Assignment 0 is due this Friday half an hour before the start of class (1:00PM Pacific time). The assignment and its starter files are up on the course website.
  - No programming involved, but you'll need to get your development environment set up.
  - There's a bunch of documentation up on the course website. Please feel free to reach out to us if there's anything we can do to help out!

Next Time

- ***Welcome to C++!***
  - Defining functions.
  - Basic arithmetic.
  - Writing loops.