Welcome to CS106B!

Who's Here Today?

- Aero/Astro
- African/Afro-American Studies
- Anthropology
- Applied Physics
- Bioengineering
- Biology
- Business
- CME
- Cancer Biology
- Chemistry
- Chinese
- CEE
- Computer Science
- Economics
- EE
- Energy Resources Engineering
- Engineering
- Environmental Systems Engineering
- Film and Media Studies
- Geophysics
- Human Biology
- International Policy
- IR
- Law
- MCS
- MS&E
- Materials Science and Engineering
- Mathematics
- MechE
- Medicine
- Music
- Philosophy
- Public Policy
- STS
- Sociology
- Statistics
- Structural Biology
- Symbolic Systems
- ***Undeclared!***
- Urban Studies

Course Staff

Keith Schwarz
htiek@cs.stanford.edu

Neel Kishnani
neelk@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/32194/

- Next, find the pinned thread at the top entitled

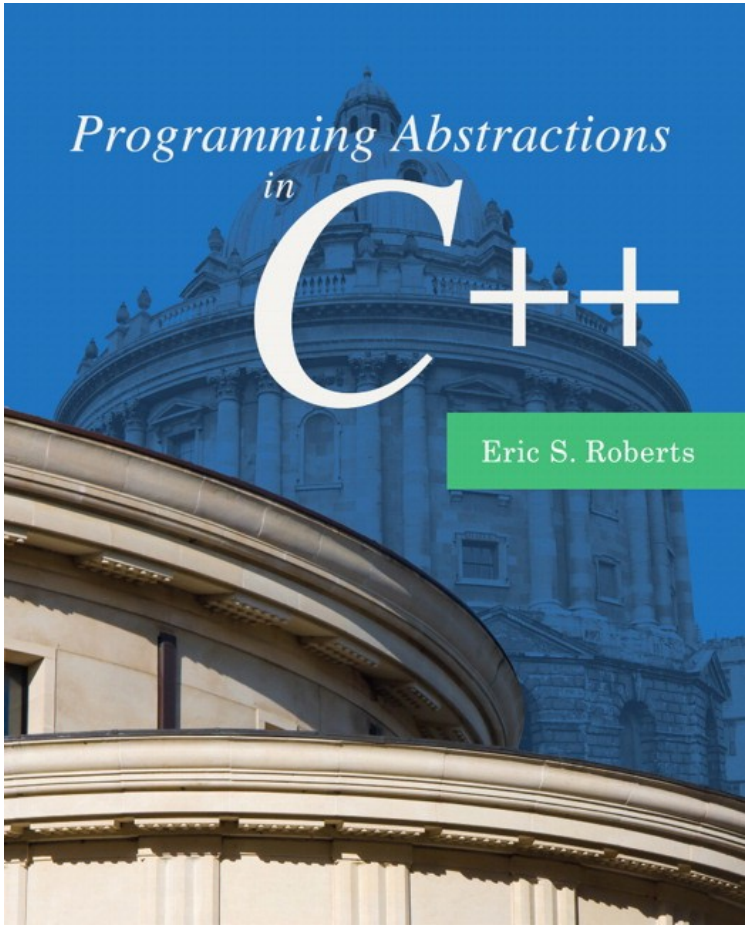
  L00: Introduction

- 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.

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 12th at 5:00PM Pacific and Sunday, January 15th 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.
  - CS100B 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

Midterm Exam

Monday, February 13th
7PM – 10PM

Grading Policies

- 40% Assignments
- 20% Midterm Exam
- 30% Final Exam

Final Exam

Monday, March 20th
8:30AM – 11:30AM

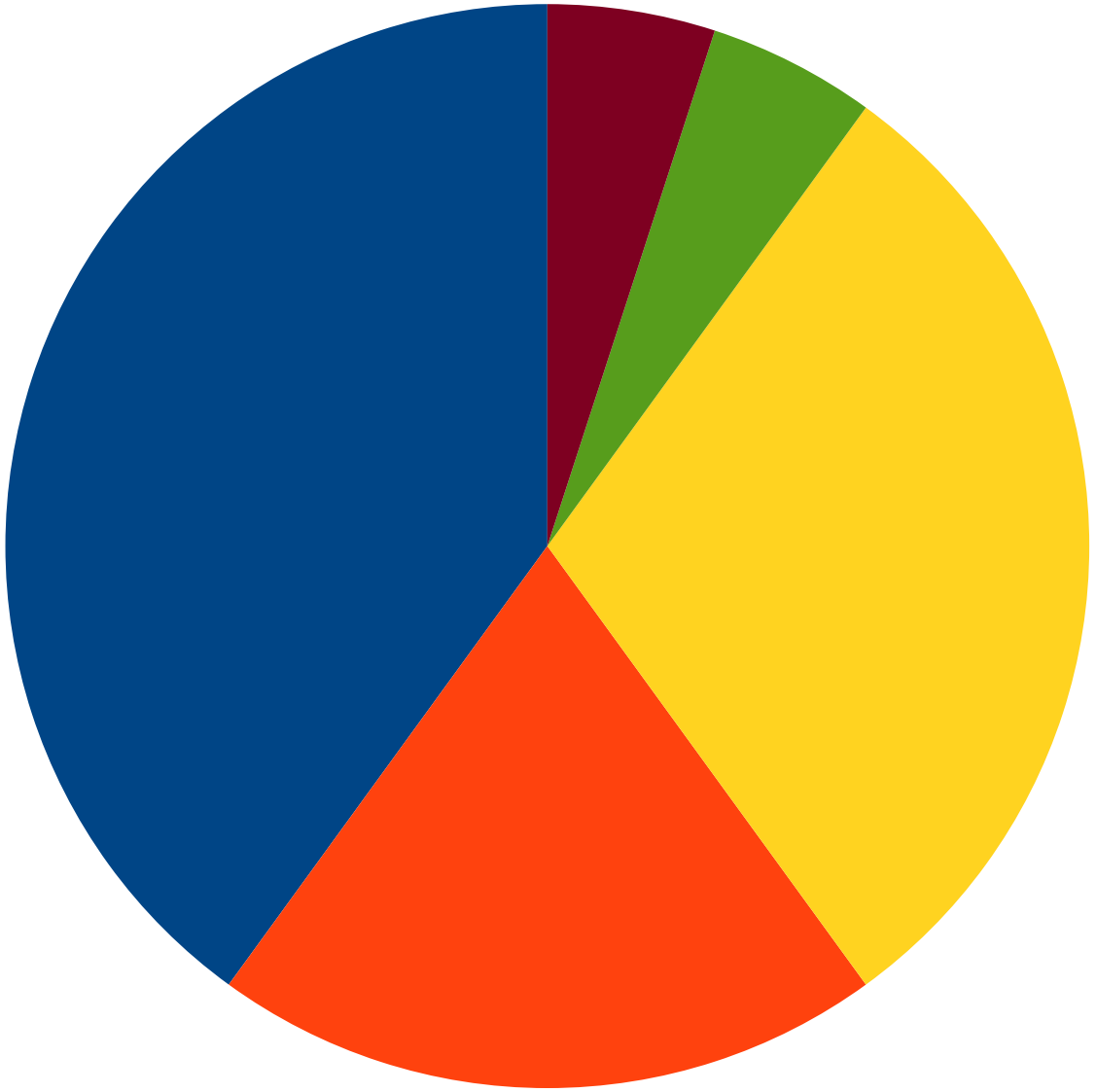
Grading Policies

- 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

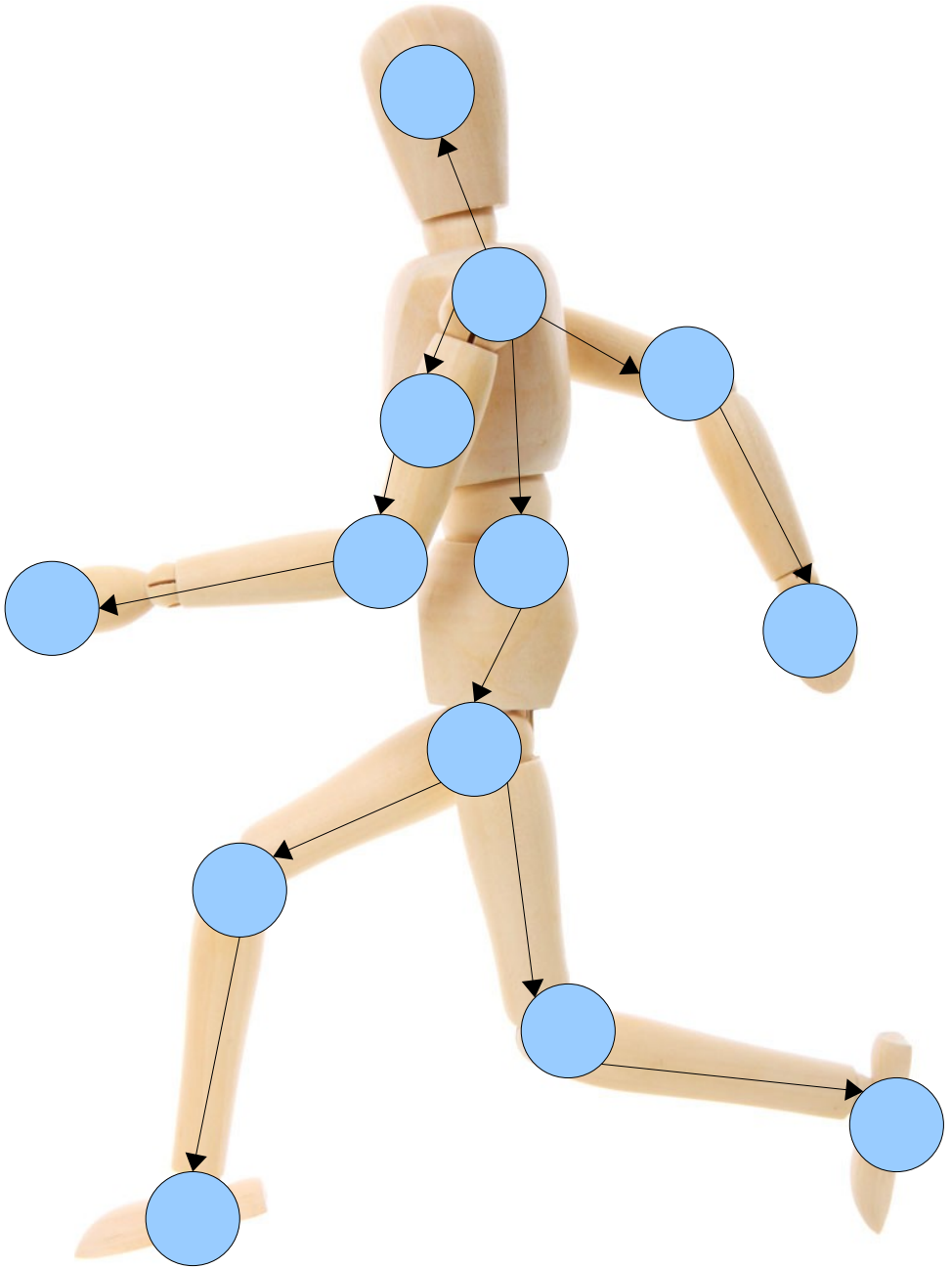
Learn how to model and solve complex problems with computers.

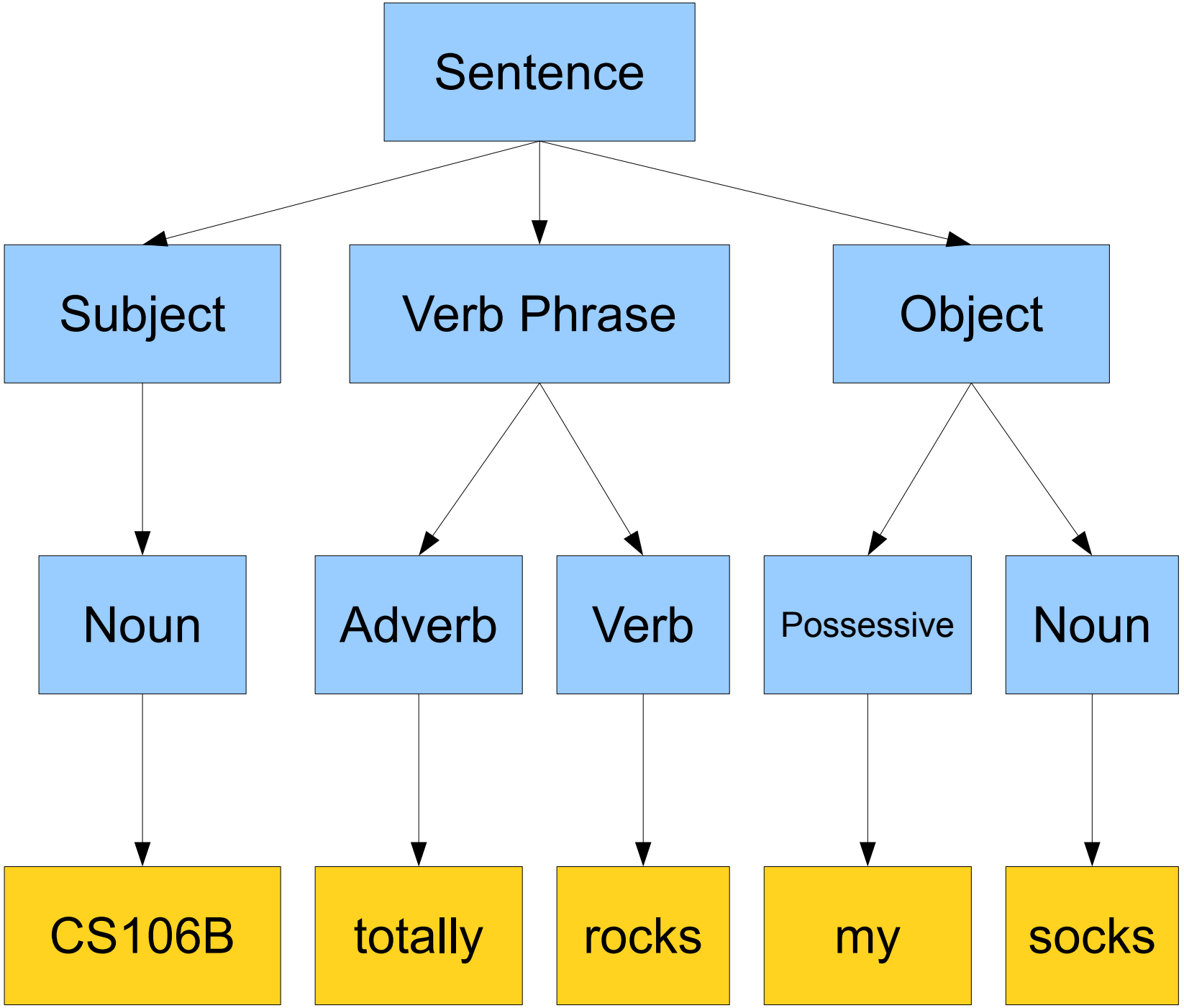
To that end:

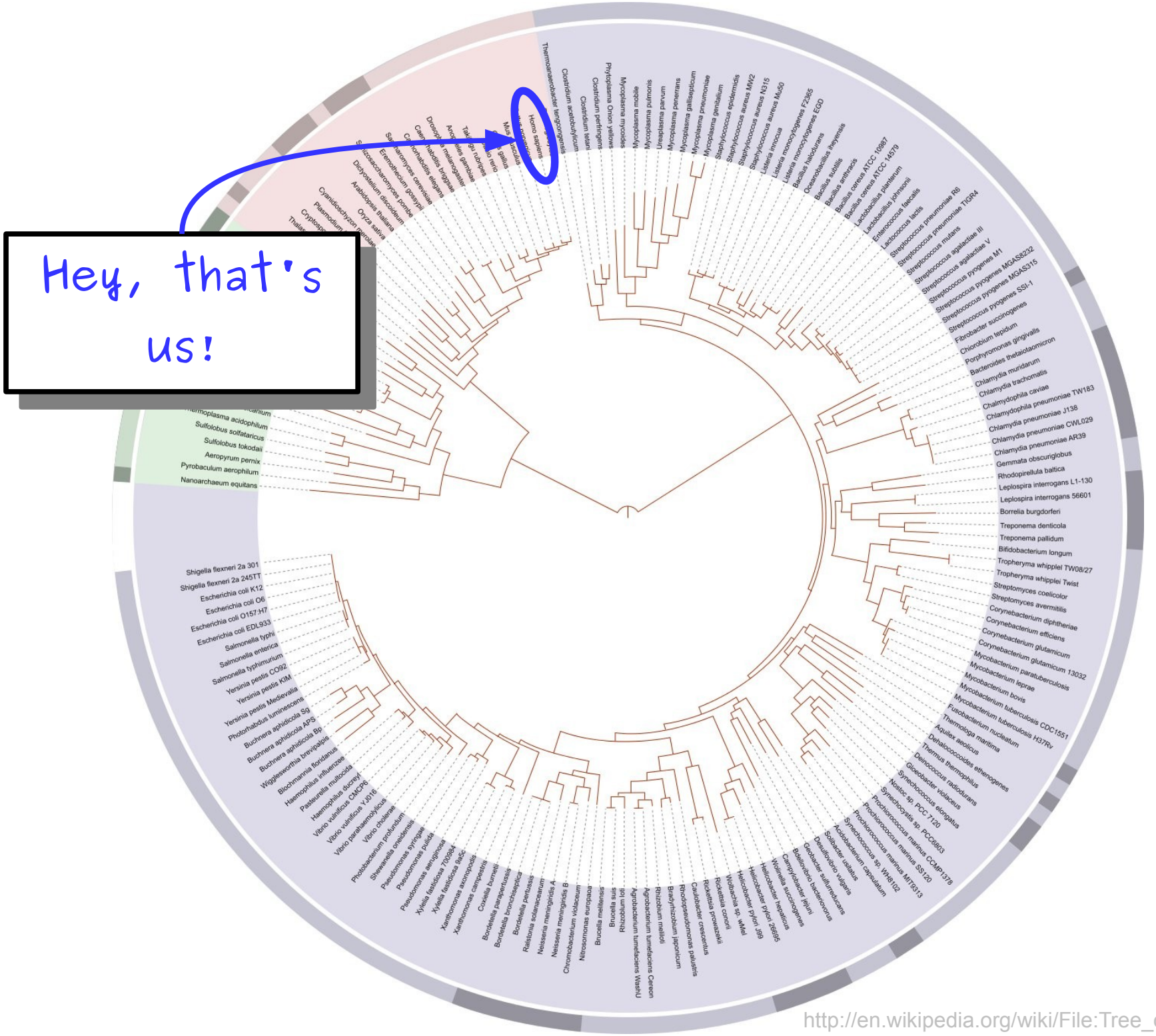
- Explore common abstractions for representing problems.

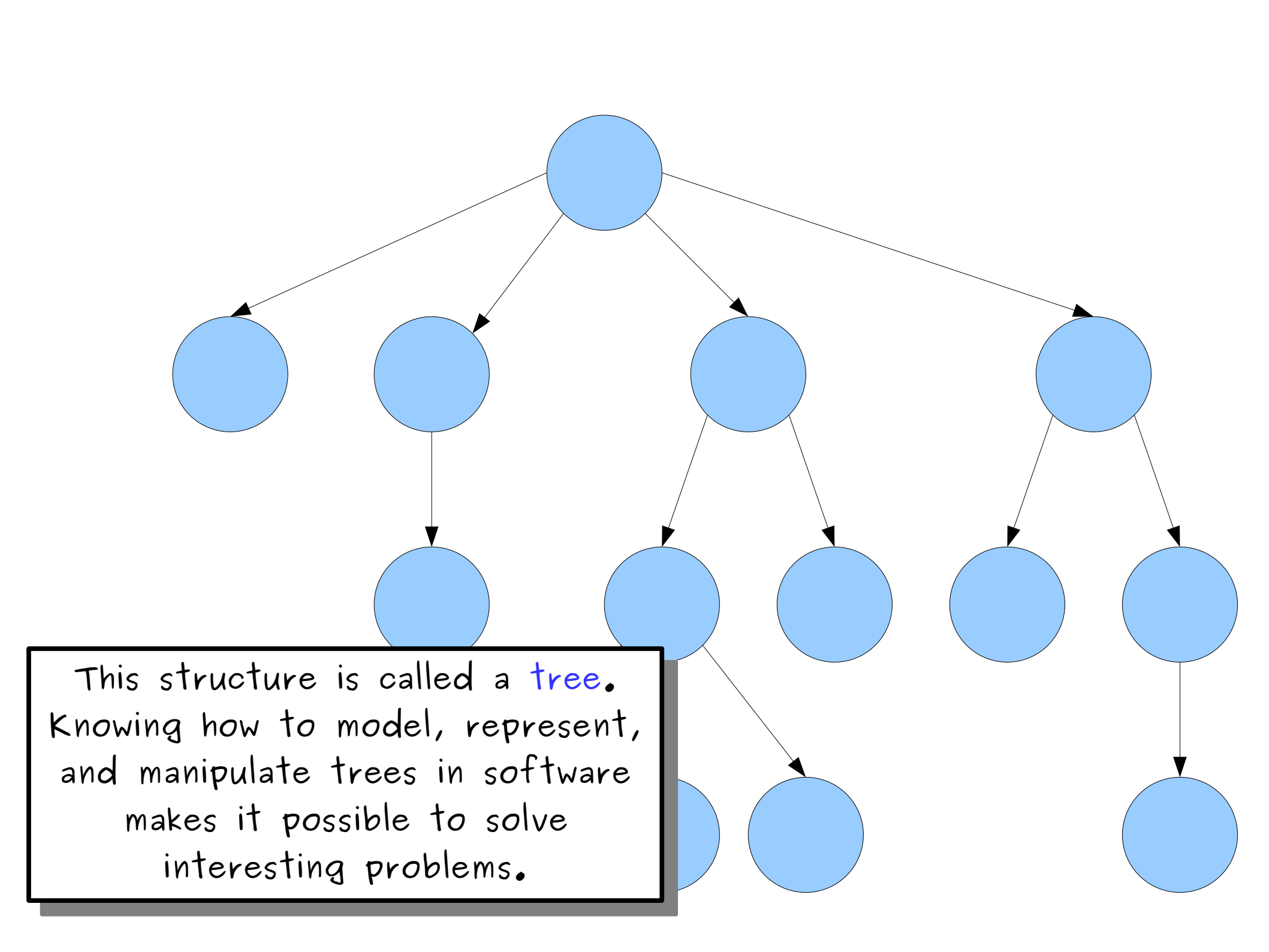
  Harness recursion and understand how to think about problems recursively.

  Quantitatively analyze different approaches for solving problems.




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




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

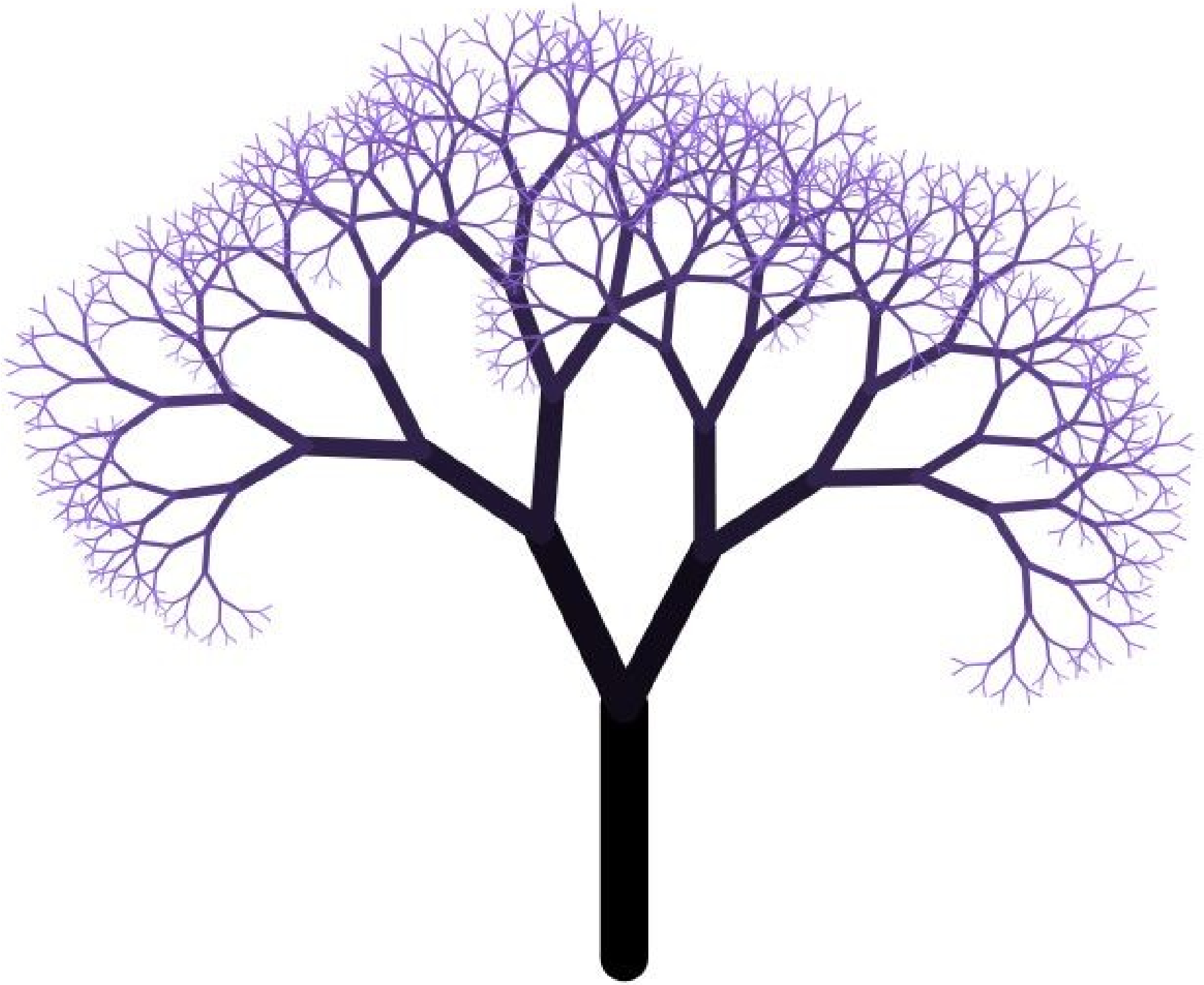
Learn how to model and solve complex problems with computers.

To that end:

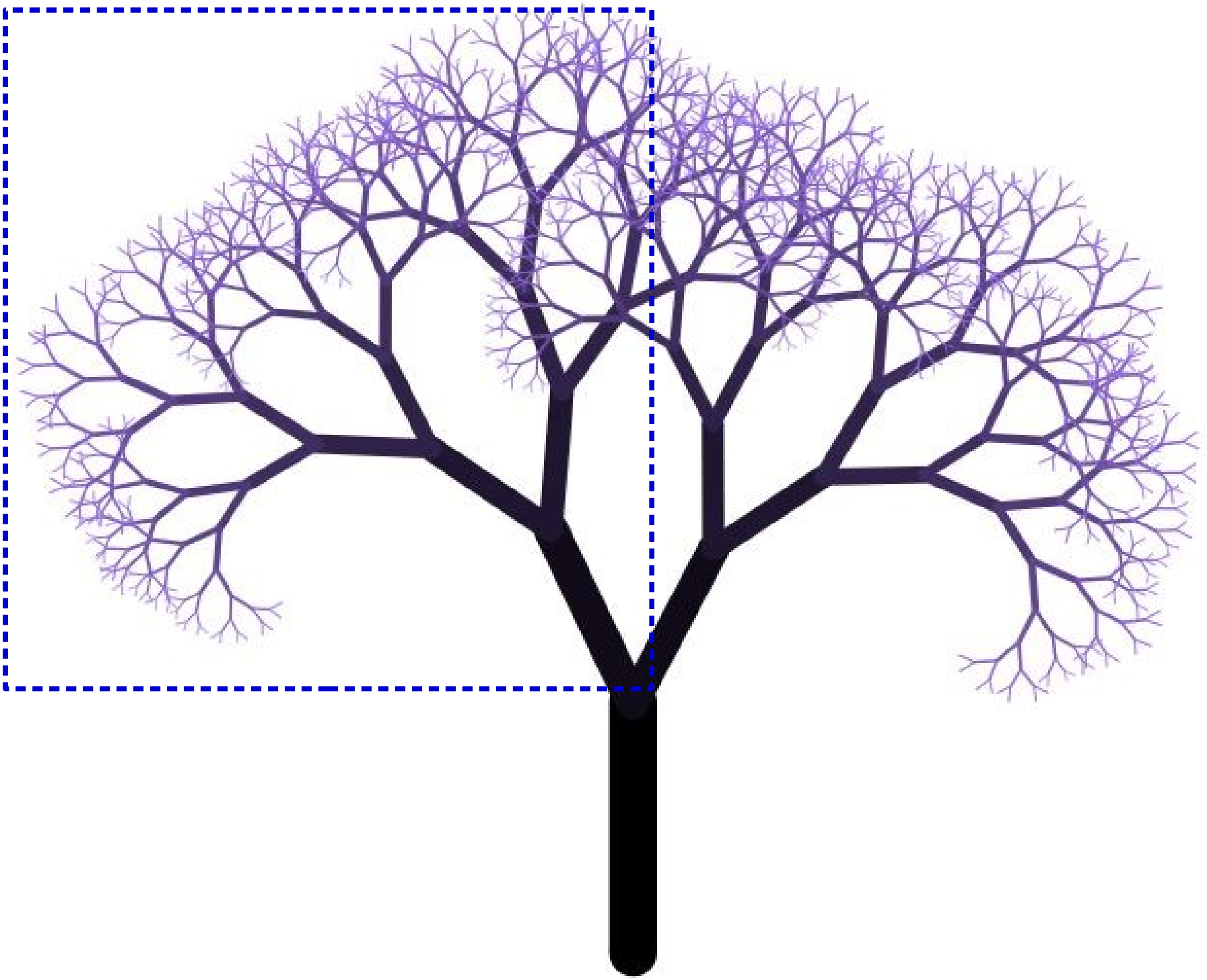
Explore common abstractions for representing problems.

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

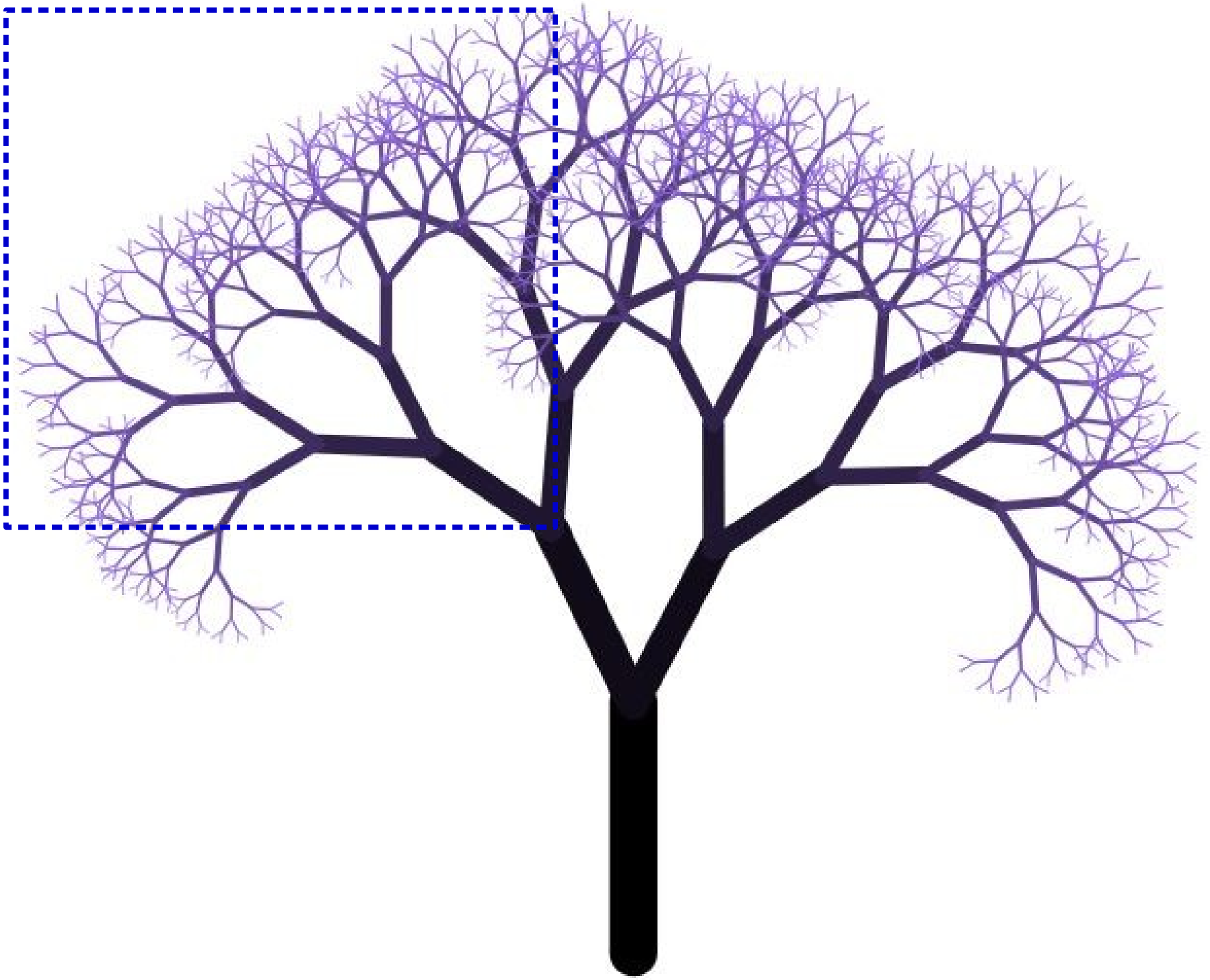
Quantitatively analyze different approaches for solving problems.



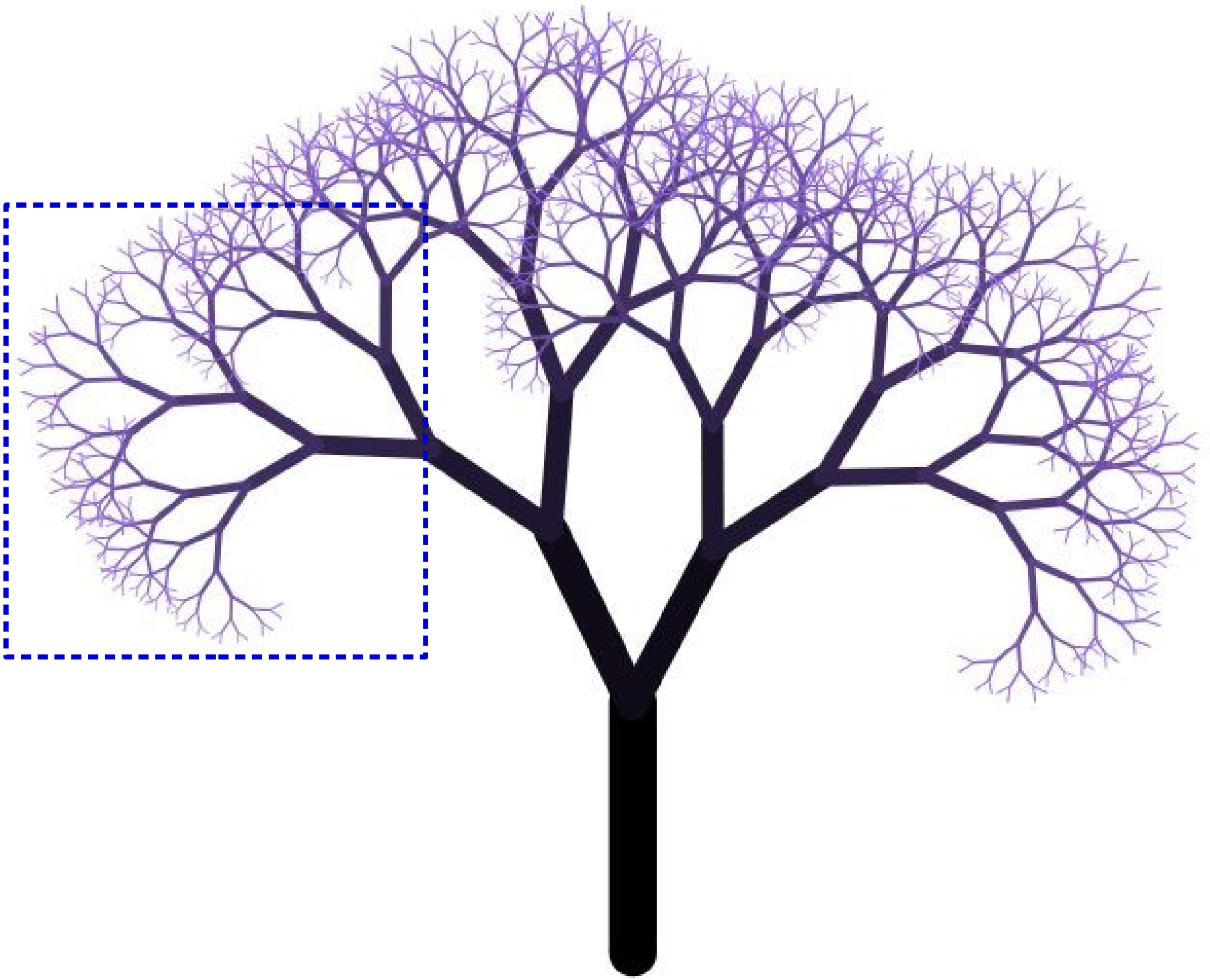
http://www.marketoracle.co.uk/images/2010/Oct/fractal-tree2.jpg



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?

- Aero/Astro
- African/Afro-American Studies
- Anthropology
- Applied Physics
- Bioengineering
- Biology
- Business
- CME
- Cancer Biology
- Chemistry
- Chinese
- CEE
- Computer Science
- Economics
- EE
- Energy Resources Engineering
- Engineering
- Environmental Systems Engineering
- Film and Media Studies
- Geophysics
- Human Biology
- International Policy
- IR
- Law
- MCS
- MS&E
- Materials Science and Engineering
- Mathematics
- MechE
- Medicine
- Music
- Philosophy
- Public Policy
- STS
- Sociology
- Statistics
- Structural Biology
- Symbolic Systems
- ***Undeclared!***
- Urban Studies

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

Perfect Numbers

- A positive integer n is called a ***perfect number*** if it's equal to the sum of its positive divisors (excluding itself).
- For example:
  - 6 is perfect since 1, 2, and 3 divide 6 and $1 + 2 + 3 = 6$.
  - 28 is perfect since 1, 2, 4, 7, and 14 divide 28 and $1 + 2 + 4 + 7 + 14 = 28$.
  - 35 isn't perfect, since 1, 5, and 7 divide 35 and $1 + 5 + 7 \neq 35$.
- Let's find the first four perfect numbers.

```
def sumOfDivisorsOf(n):
    """Returns the sum of the positive divisors of the number n >= 0."""
    total = 0

    for i in range(1, n):
        if n % i == 0:
            total += i



    return total;



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

# Keep looking until we've found four perfect numbers.
while found < 4:
    # A number is perfect if the sum of its divisors is equal to it.
    if sumOfDivisorsOf(number) == number:
        print(number)
        found += 1


    number += 1
```

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

/* Returns the sum of the positive divisors of the number n >= 0. */
int sumOfDivisorsOf(int n) {
    int total = 0;

    for (int i = 1; i < n; i++) {
        if (n % i == 0) {
            total += i;
        }
    }

    return total;
}

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

    /* Keep looking until we've found four perfect numbers. */
    while (found < 4) {
        /* A number is perfect if the sum of its divisors is equal to it. */
        if (sumOfDivisorsOf(number) == number) {
            cout << number << endl;
            found++;
        }

        number++;
    }

    return 0;
}
```

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

/* Returns the sum of the positive divisors of the number n >= 0. */
int sumOfDivisorsOf(int n) {
    int total = 0;

    for (int i = 1; i < n; i++) {
        if (n % i == 0) {
            total += i;
        }
    }

    return total;
}

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

    /* Keep looking until we've found four perfect numbers. */
    while (found < 4) {
        /* A number is perfect if the sum of its divisors is equal to it. */
        if (sumOfDivisorsOf(number) == number) {
            cout << number << endl;
            found++;
        }

        number++;
    }

    return 0;
}
```

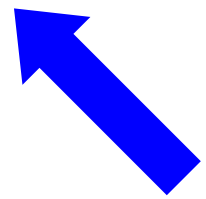
In Python, indentation alone determines nesting.

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

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

/* Returns the sum of the positive divisors of the number n >= 0. */
int sumOfDivisorsOf(int n) {
    int total = 0;

    for (int i = 1; i < n; i++) {
        if (n % i == 0) {
            total += i;
        }
    }

    return total;
}

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

    /* Keep looking unti                                   s. */
    while (found < 4) {
        /* A number is p                                   is equal to it. */
        if (sumOfDivisors                                   {
            cout << number << endl;
            found++;
        }

        number++;
    }

    return 0;
}
```

In Python, newlines mark the end of statements.

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



```
total += i;
```

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

/* Returns the sum of the positive divisors of the number n >= 0. */
int sumOfDivisorsOf(int n) {
    int total = 0;

    for (int i = 1; i < n; i++) {
        if (n % i == 0) {
            total += i;
        }
    }

    return total;
}

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

    /* Keep looking until we've found four perfect numbers. */
    while (found < 4) {
        /* A number is perfect if the sum of its divisors is equal to it. */
        if (sumOfDivisorsOf(number) == number) {
            cout << number << endl;
            found++;
        }

        number++;
    }

    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 the sum of the positive divisors of the number n >= 0. */
int sumOfDivisorsOf(int n) {
    int total = 0;

    for (int i = 1; i < n; i++) {
        if (n % i == 0) {
            total += i;
        }
    }

    return total;
}

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

    /* Keep looking until we've found four perfect numbers. */
    while (found < 4) {
        /* A number is perfect if the sum of its divisors is equal to it. */
        if (sumOfDivisorsOf(number) == number) {
            cout << number << endl;
            found++;
        }

        number++;
    }

    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 the sum of the positive divisors of the number n >= 0. */
int sumOfDivisorsOf(int n) {
    int total = 0;

    for (int i = 1; i < n; i++) {
        if (n % i == 0) {
            total += i;
        }
    }

    return total;
}

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

    /* Keep looking until we've found four perfect numbers. */
    while (found < 4) {
        /* A number is perfect if the sum of its divisors is equal to it. */
        if (sumOfDivisorsOf(number) == number) {
            cout << number << endl;
            found++;
        }

        number++;
    }

    return 0;
}
```

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

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

/* Returns the sum of the positive divisors of the number n >= 0. */
int sumOfDivisorsOf(int n) {
    int total = 0;

    for (int i = 1; i < n; i++) {
        if (n % i == 0) {
            total += i;
        }
    }

    return total;
}

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

    /* Keep looking until we've found four perfect numbers. */
    while (found < 4) {
        /* A number is perfect if the sum of its divisors is equal to it. */
        if (sumOfDivisorsOf(number) == number) {
            cout << number << endl;
            found++;
        }

        number++;
    }

    return 0;
}
```

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

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

/* Returns the sum of the positive divisors of the number n >= 0. */
int sumOfDivisorsOf(int n) {
    int total = 0;

    for (int i = 1; i < n; i++) {
        if (n % i == 0) {
            total += i;
        }
    }

    return total;
}

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

    /* Keep looking until we've found four perfect numbers. */
    while (found < 4) {
        /* A number is perfect if the sum of its divisors is equal to it. */
        if (sumOfDivisorsOf(number) == number) {
            cout << number << endl;
            found++;
        }

        number++;
    }

    return 0;
}
```

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

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

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

/* Returns the sum of the positive divisors of the number n >= 0. */
int sumOfDivisorsOf(int n) {
    int total = 0;

    for (int i = 1; i < n; i++) {
        if (n % i == 0) {
            total += i;
        }
    }

    return total;
}

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

    /* Keep looking until we've found four perfect numbers. */
    while (found < 4) {
        /* A number is perfect if the sum of its divisors is equal to it. */
        if (sumOfDivisorsOf(number) == number) {
            cout << number << endl;
            found++;
        }

        number++;
    }

    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.)

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

/* Returns the sum of the positive divisors of the number n >= 0. */
int sumOfDivisorsOf(int n) {
    int total = 0;

    for (int i = 1; i < n; i++) {
        if (n % i == 0) {
            total += i;
        }
    }

    return total;
}

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

    /* Keep looking until we've found four perfect numbers. */
    while (found < 4) {
        /* A number is perfect if the sum of its divisors is equal to it. */
        if (sumOfDivisorsOf(number) == number) {
            cout << number << endl;
            found++;
        }

        number++;
    }

    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.