Welcome to CS106B!

- Visit the course website at

  https://cs106b.stanford.edu

  for access to materials for today:
  - Course Syllabus
  - Course Calendar
  - Course Placement Info
  - Honor Code Policies
  - Assignment 0

Who's Here Today?

- Aero/Astro
- Afro-American Studies
- Anthropology
- Art History
- Biochemistry
- Bioengineering
- Biology
- Biomedical Informatics
- Business
- Chemistry
- Civil/Env. Engr
- Classics
- Creative Writing
- Comparative Lit
- CSRE
- Computer Science
- CME
- Earth Systems
- Economics
- Education
- Electrical Engineering
- Energy Resources
- Epidemiology
- Human Biology
- Immunology
- International Policy
- Intl. Relations
- Latin Amer. Studies
- Law
- Mech. Engineering
- MS&E
- Neuroscience
- Physics
- Psychology
- Public Policy
- Statistics
- TAPS
- ***Undeclared!***
- Urban Studies

Course Staff

Instructor: Keith Schwarz
(htiek@cs.stanford.edu)

Head TA: Neel Kishnani
(neelk@stanford.edu)

The CS106B Section Leaders
The CS106B Course Helpers

Asking Questions

- We've set up an online system you can use to ask us questions in lecture.
- First, visit our EdStem page. It's linked through the course Canvas and also available here:

https://edstem.org/us/courses/16604/

- 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.
- Post any questions as a response to this thread. The course staff will respond to questions as they come in. I'll periodically take time out of lecture to go over some of the more popular ones.

Course Website

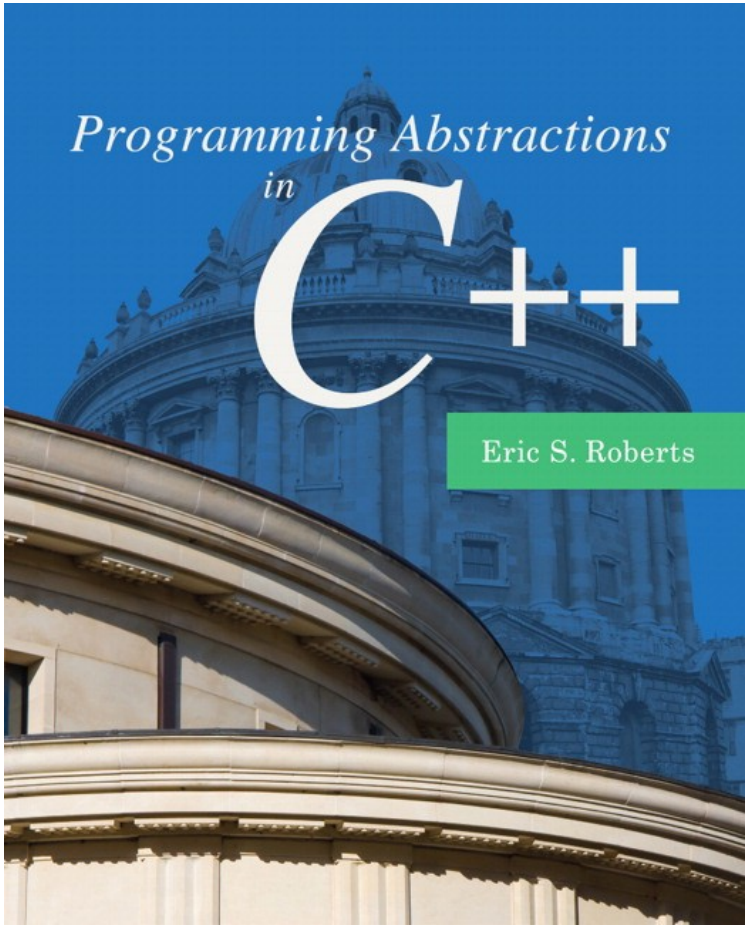
https://cs106b.stanford.edu

Prerequisites

CS106A

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

Textbook Options



- The course textbook has excellent explanations of course topics and is a great reference for C++ as we'll use it in this course.
- There's also a **draft version** available online that you can use this quarter.

Grading Policies

Grading Policies

40% Assignments

Ten Assignments

(One intro assignment that goes out today, nine programming assignments)

Grading Policies

Midterm Exam

Goes out Friday, February 4th
Due Sunday, February 6th

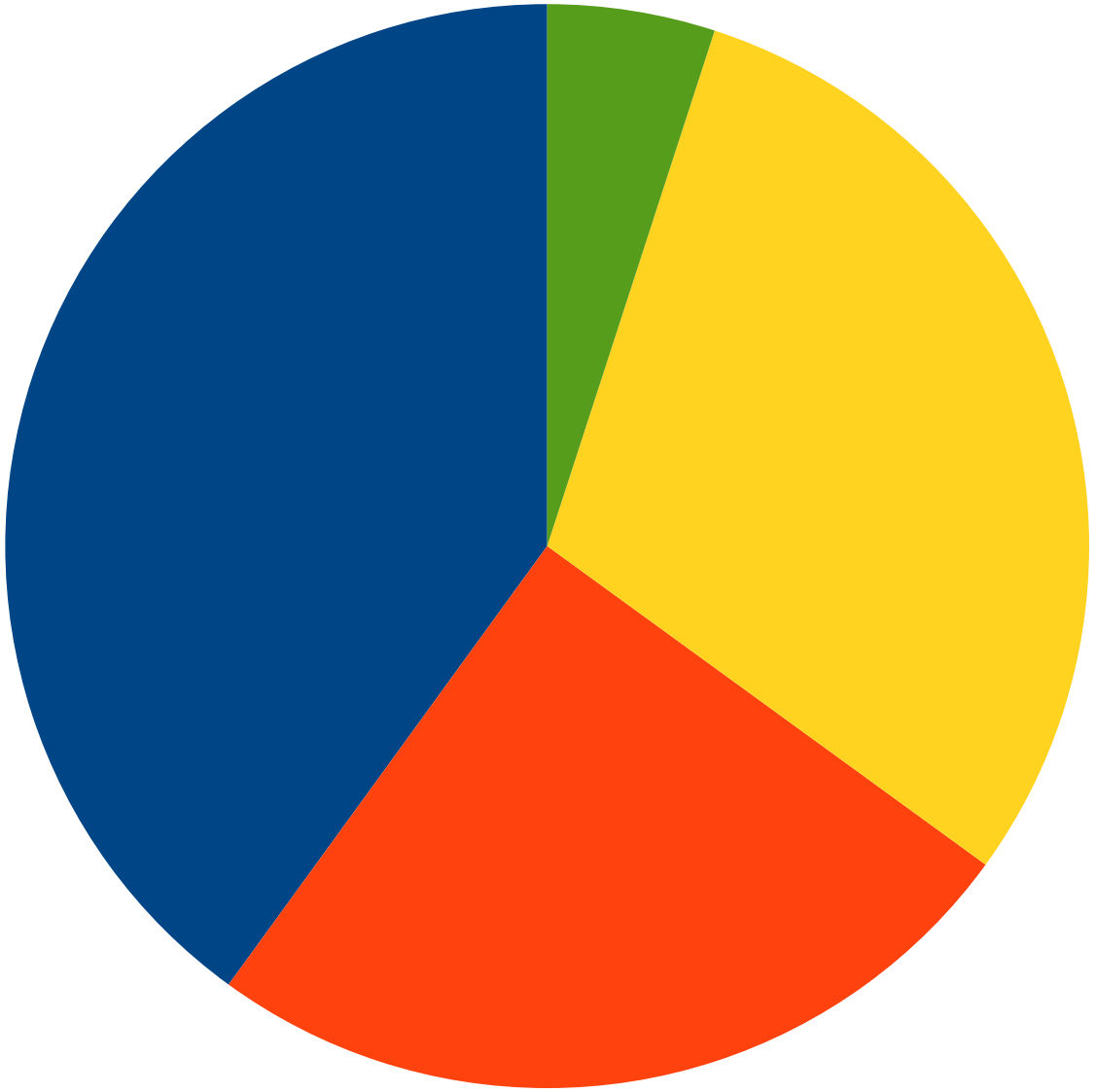
Grading Policies

- 40% Assignments
- 25% Midterm Exam
- 30% Final Exam

Final Exam

Goes out Friday, March 11th
Due Monday, March 14th

Grading Policies



- 40% Assignments
- 25% Midterm Exam
- 30% Final Exam
- 5% Section Participation

Discussion Sections

Weekly sections.
Let’s talk about them!

Discussion Sections

- There are weekly discussion sections in CS106B. Section attendance is required.
- Sign up between Thursday, January 6th at 5:00PM Pacific and Sunday, January 9th at 5:00PM Pacific by visiting

  https://cs198.stanford.edu/cs198/auth/default.aspx
- We don't look at Axess for section enrollments. Please make sure to sign up here even if you're already enrolled on Axess.
- 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.

CS100B

- CS100B is an optional, one-unit add-on to CS106B that provides extra practice with the material.
  - It's run in addition to, rather than in place of, the normal CS106B weekly discussion sections.
- It's run through the School of Engineering's ACE program. The application is available online here:

  https://forms.gle/WwhfG7Zdyhpa8Gi97

- Questions? Contact Breauna Spencer at **bspence2@stanford.edu**.

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

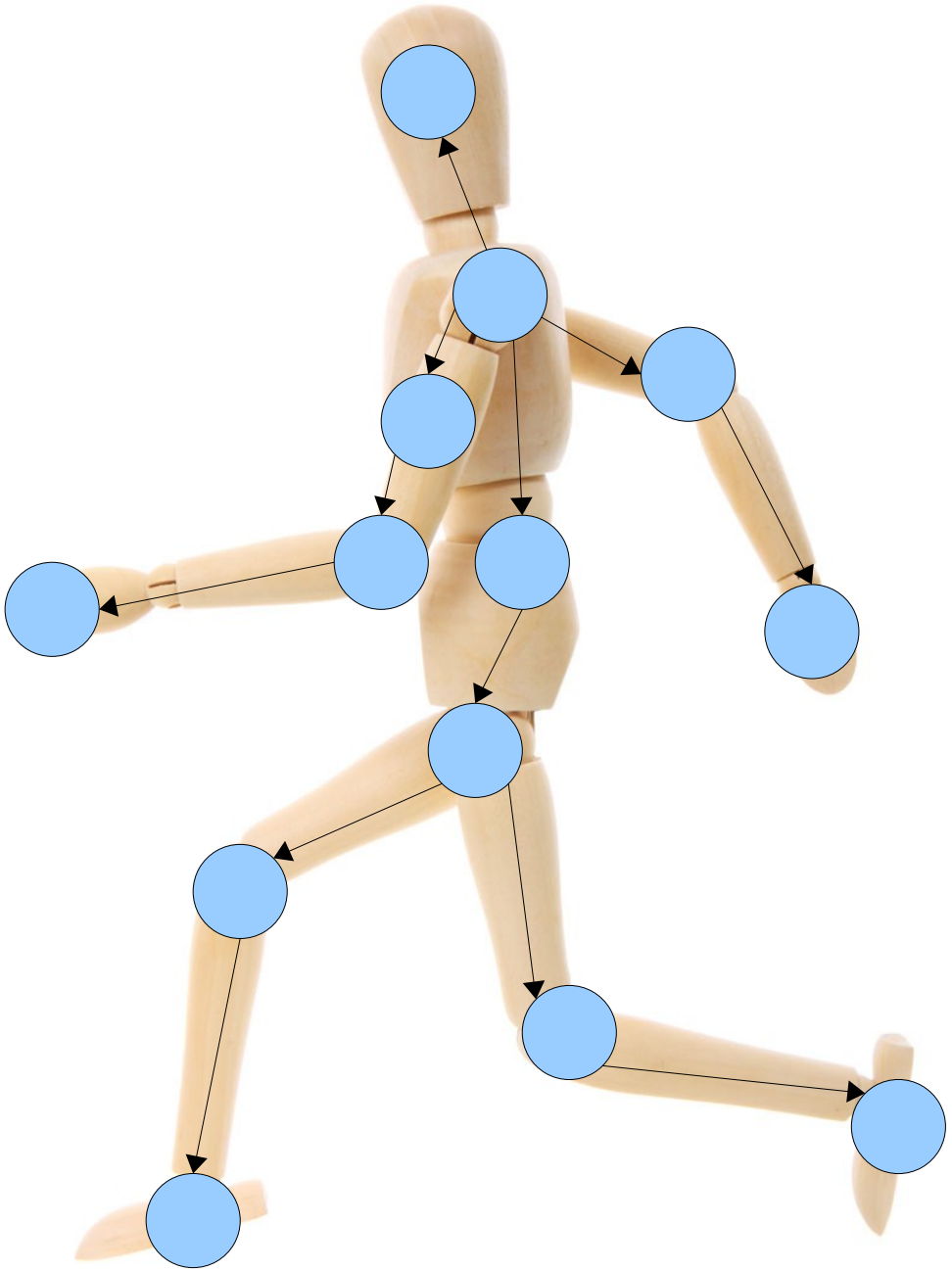
Learn how to model and solve complex problems with computers.

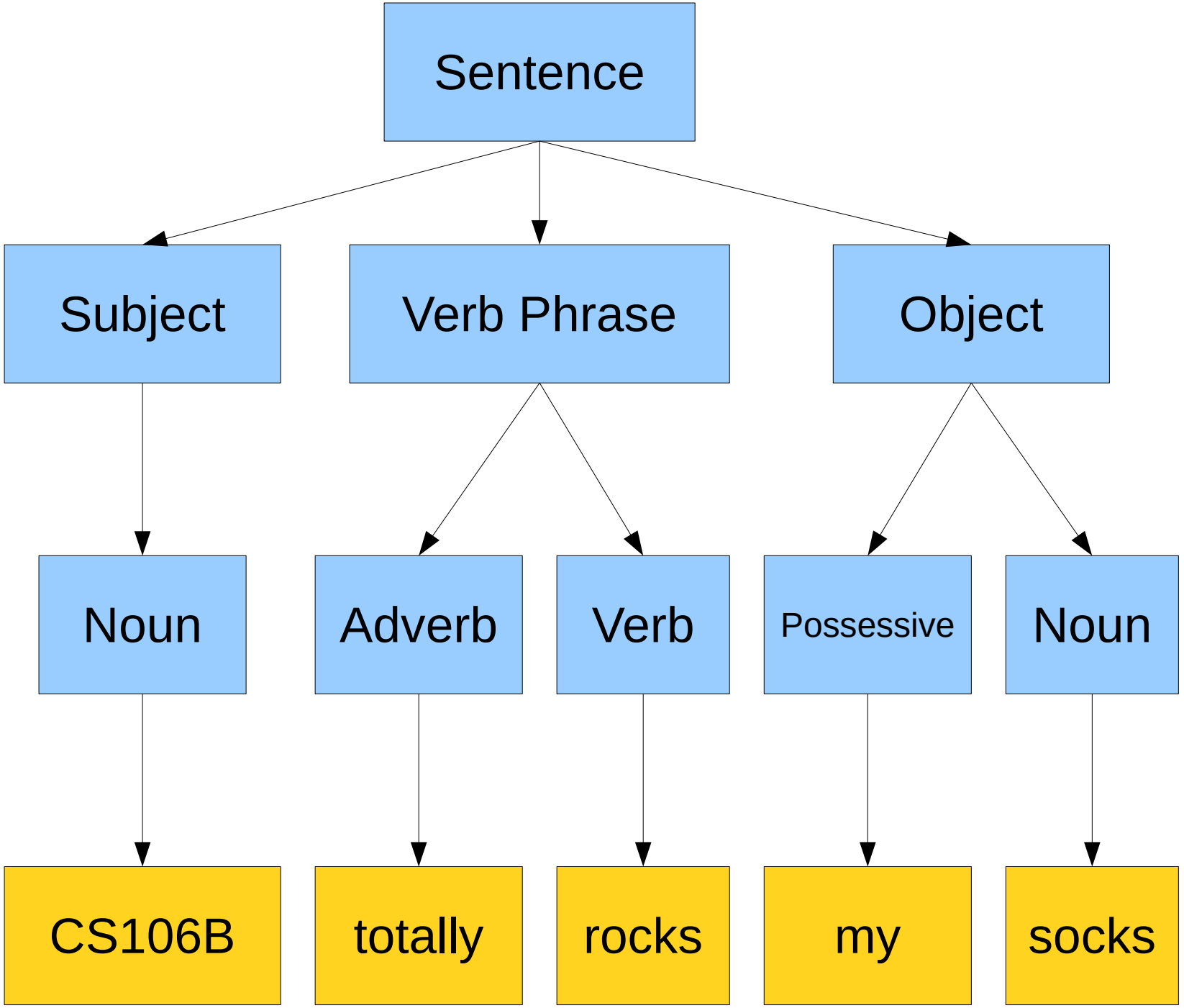
To that end:

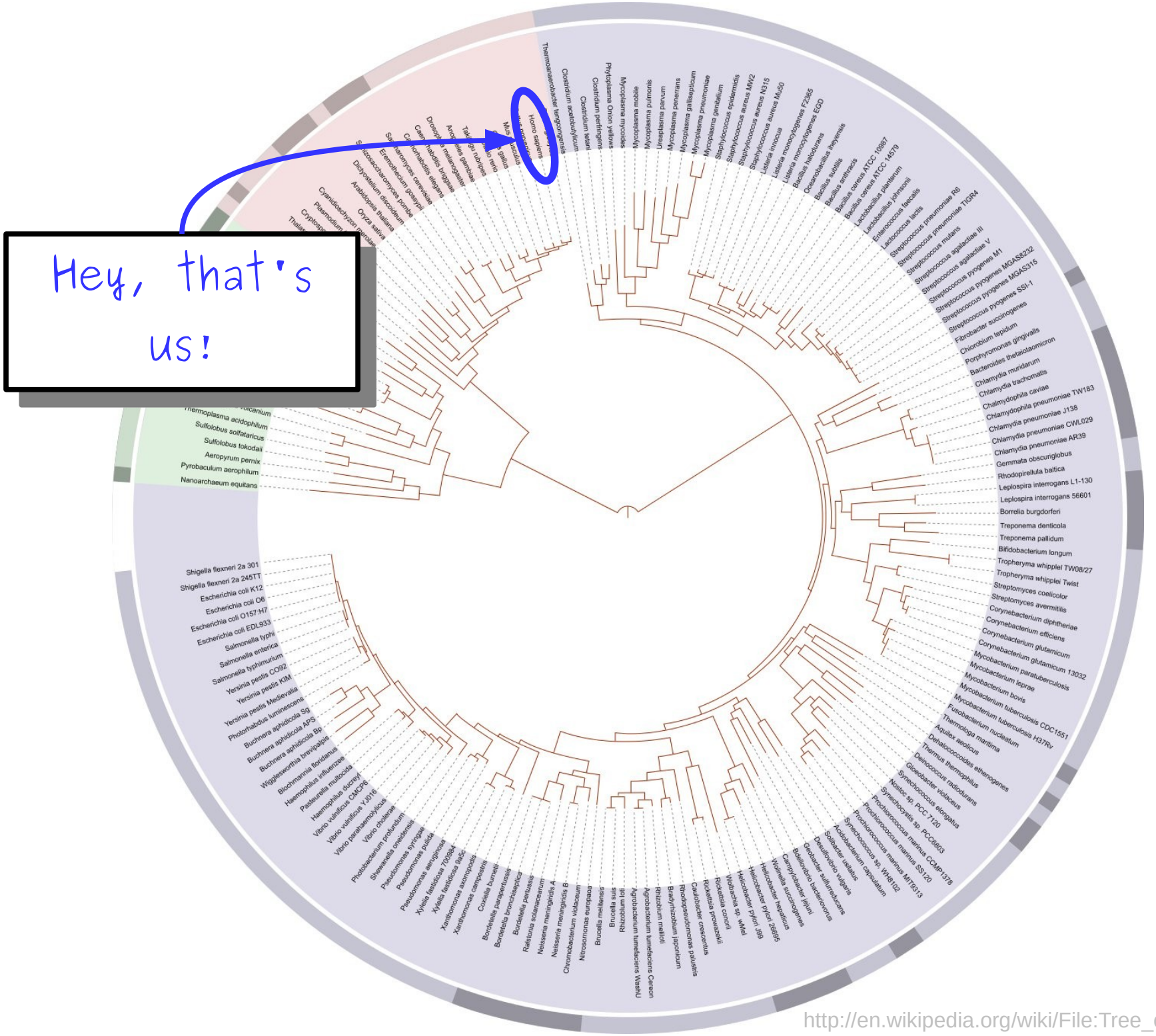
- Explore common abstractions for representing problems.

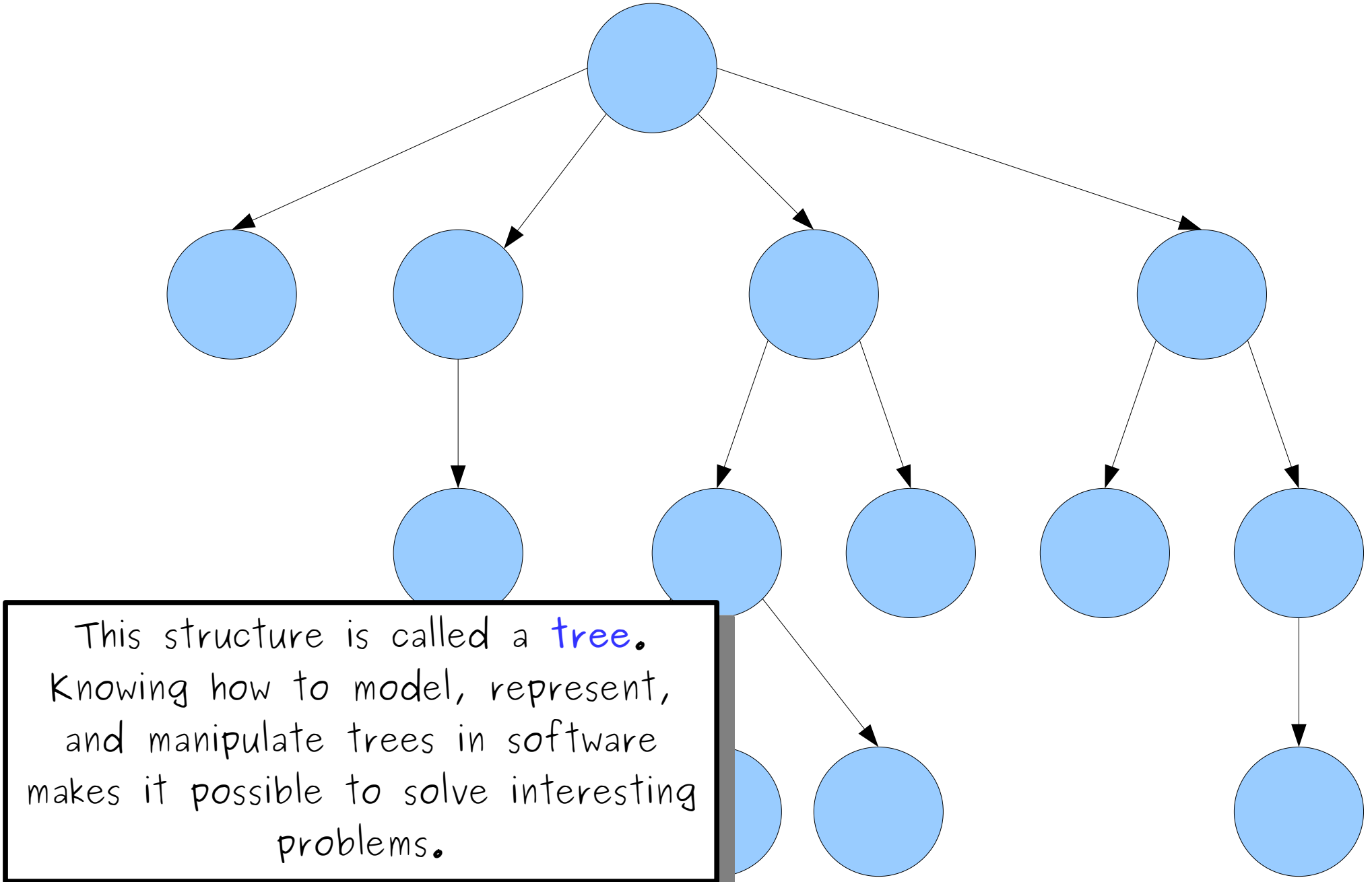
  Harness recursion and understand how to think about problems recursively.

  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

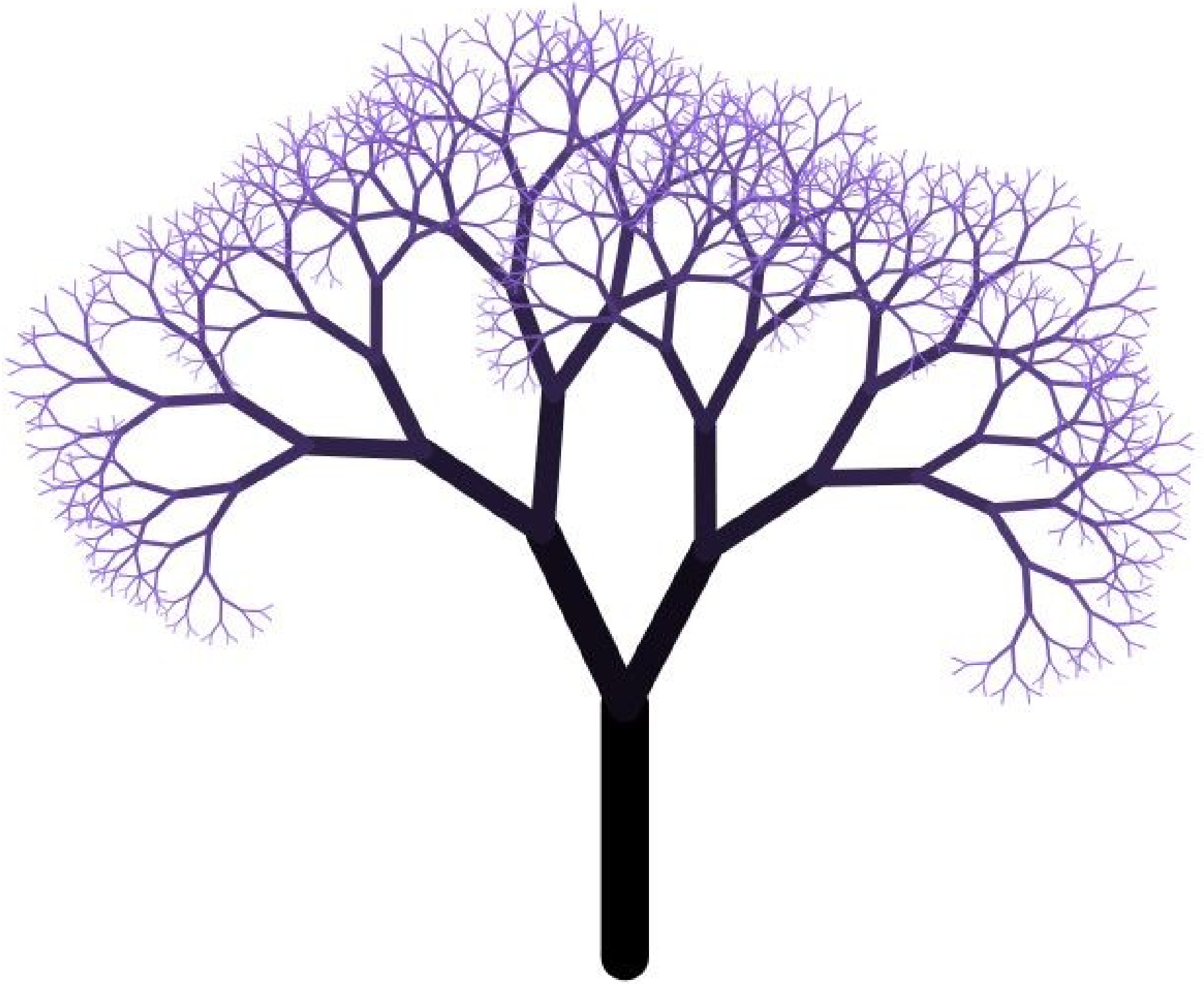
Learn how to model and solve complex problems with computers.

To that end:

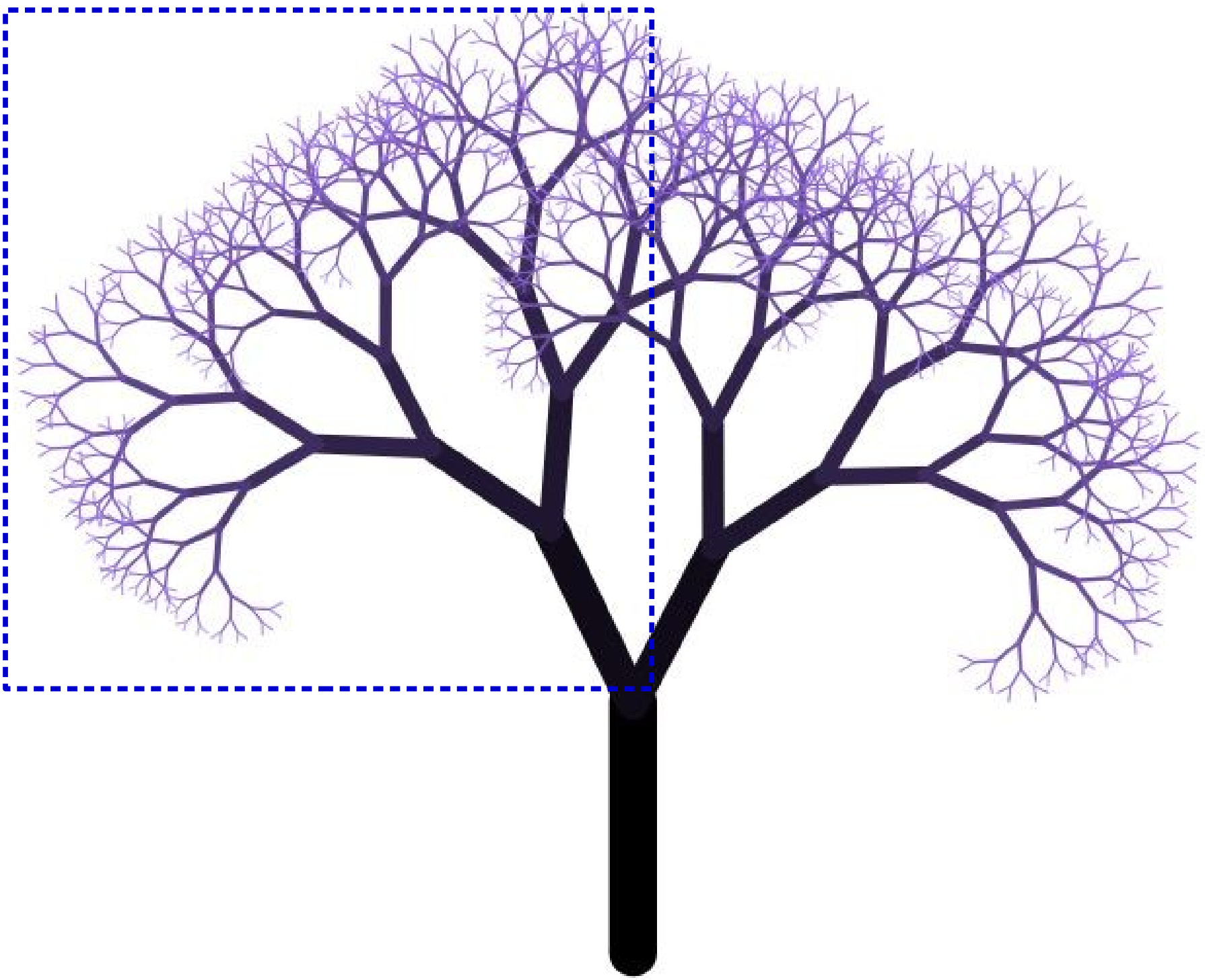
Explore common abstractions for representing problems.

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

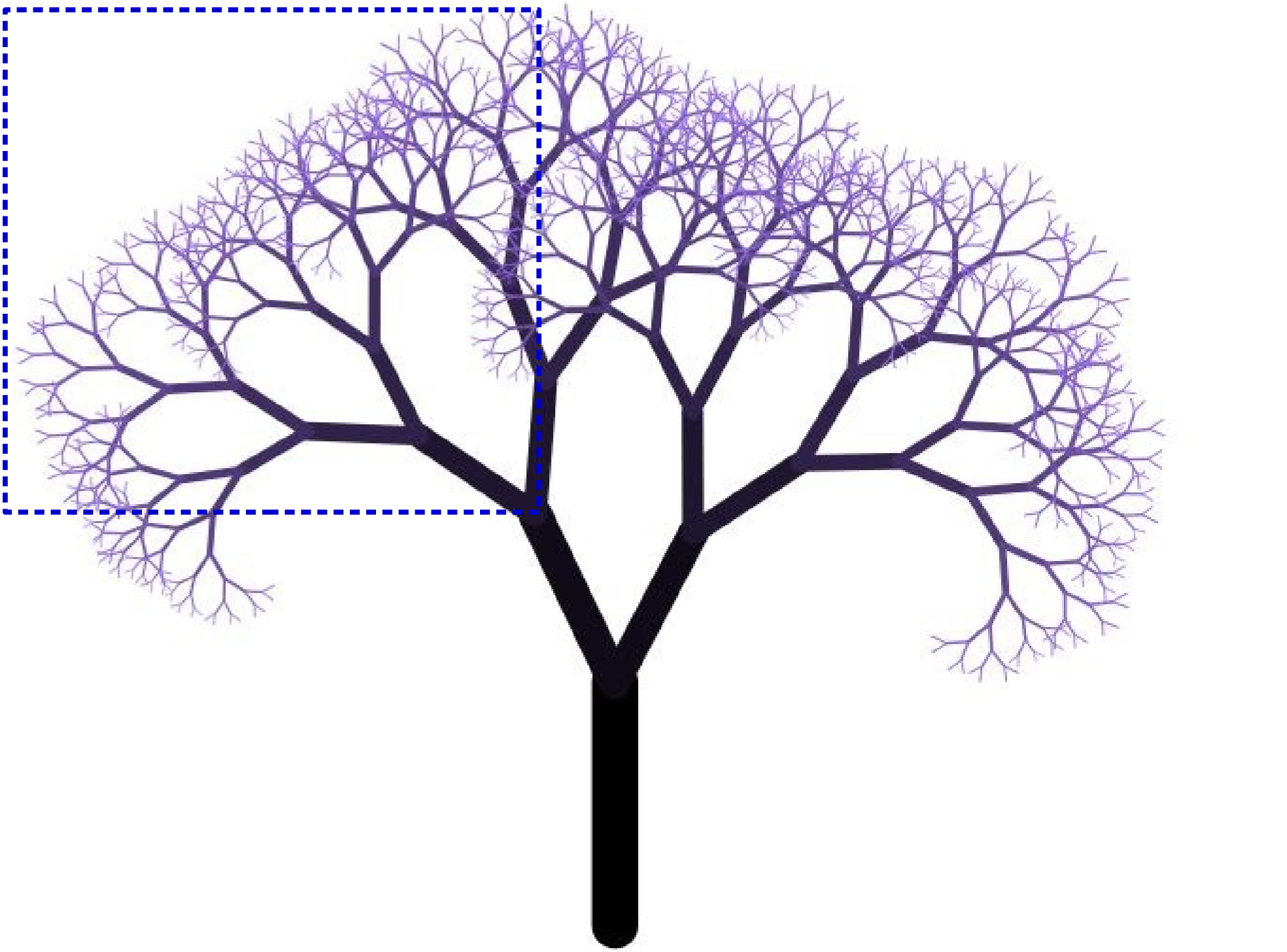
Quantitatively analyze different approaches for solving problems.



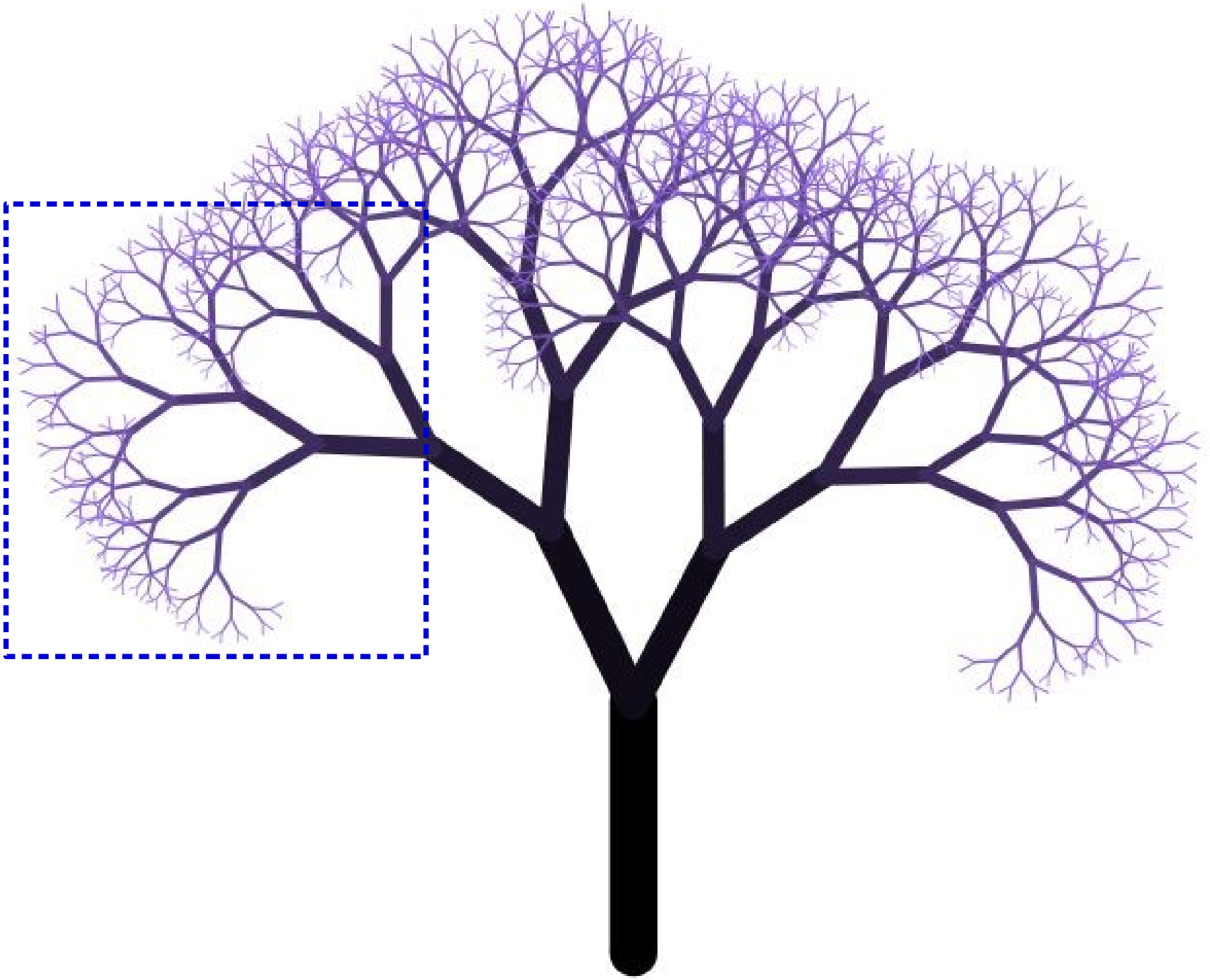
http://www.marketoracle.co.uk/images/2010/Oct/fractal-tree2.jpg



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

Source: https://datacenterfrontier.com/year-hyperscale-facebook-growth-innovation/

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
- Afro-American Studies
- Anthropology
- Art History
- Biochemistry
- Bioengineering
- Biology
- Biomedical Informatics
- Business
- Chemistry
- Civil/Env. Engr
- Classics
- Creative Writing
- Comparative Lit
- CSRE
- Computer Science
- CME
- Earth Systems
- Economics
- Education
- Electrical Engineering
- Energy Resources
- Epidemiology
- Human Biology
- Immunology
- International Policy
- Intl. Relations
- Latin Amer. Studies
- Law
- Mech. Engineering
- MS&E
- Neuroscience
- Physics
- Psychology
- Public Policy
- Statistics
- TAPS
- ***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 way easier than learning a first. You already know how to solve problems; you just need to adjust the syntax you use.

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
```

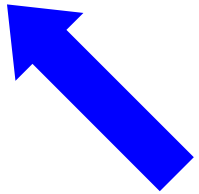
```cpp
#include <iostream>
using namespace std;

/* 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;  // 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, newlines mark the end of statements.

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

```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++, all 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.

C++: The Basics

```cpp
/*            C++ Version                */
double areaOfCircle(double r) {
    return M_PI * r * r;
}

int maxOf(int first, int second) {
    if (first > second) {
        return first;
    }
    return second;
}

void printNumber(int n) {
    cout << "I like " << n << endl;
}
```

```python
"""       Python Version          """
def areaOfCircle(r):
    return math.pi * r * r


def maxOf(first, second):
    if first > second:
        return first
    return second



def printNumber(n):
    print("I like " + str(n))
```

```java
/*            Java Version               */
private double areaOfCircle(double r) {
    return M_PI * r * r;
}

private int maxOf(int first, int second) {
    if (first > second) {
        return first;
    }
    return second;
}

private void printNumber(int n) {
    System.out.println("I like " + n);
}
```

```javascript
//      JavaScript Version
function areaOfCircle(r) {
   return Math.PI * r * r;
}

function maxOf(first, second) {
   if (first > second) {
       return first;
   }
   return second;
}

function printNumber(n) {
   console.log("I like " + n);
}
```

```cpp
/*              C++ Version                */
double areaOfCircle(double r) {
    return M_PI * r * r;
}

int maxOf(int first, int second) {
    if (first > second) {
        return first;
    }
    return second;
}

void printNumber(int n) {
    cout << "I like " << n << endl;
}
```

```python
"""        Python Version          """
def areaOfCircle(r):
    return math.pi * r * r


def maxOf(first, second):
    if first > second:
        return first
    return second



def printNumber(n):
    print("I like " + str(n))
```

```java
/*              Java Ve
private double areaOf
    return M_PI * r *
}

private int maxOf(int
    if (first > secon
        return first;
    }
    return second;
}

private void printNum
    System.out.println("I like " + n);
}
```

```javascript
    console.log("I like " + n);
}
```

Functions in C++ work like functions in Python/JavaScript or like methods in Java. They (optionally) take in parameters, perform a calculation, then (optionally) return a value.

```cpp
/*            C++ Version            */
double areaOfCircle(double r) {
    return M_PI * r * r;
}

int maxOf(int first, int second) {
    if (first > second) {
        return first;
    }
    return second;
}

void printNumber(int n) {
    cout << "I like " << n << endl;
}
```

```python
"""       Python Version       """
def areaOfCircle(r):
    return math.pi * r * r


def maxOf(first, second):
    if first > second:
        return first
    return second



def printNumber(n):
    print("I like " + str(n))
```

```java
/*            Java Ve
private double areaOf
    return M_PI * r *
}

private int maxOf(int
    if (first > secon
        return first;
    }
    return second;
}

private void printNum
    System.out.println("I like " + n);
}
```

```javascript
    console.log("I like " + n);
}
```

You define a function by writing

```
return-type fn-name(args) {
    // … code goes here …
}
```

```cpp
/*            C++ Version            */
double areaOfCircle(double r) {
    return M_PI * r * r;
}

int maxOf(int first, int second) {
    if (first > second) {
        return first;
    }
    return second;
}

void printNumber(int n) {
    cout << "I like " << n << endl;
}
```

```python
"""       Python Version       """
def areaOfCircle(r):
    return math.pi * r * r


def maxOf(first, second):
    if first > second:
        return first
    return second



def printNumber(n):
    print("I like " + str(n))
```

All variables in C++ need a type. Some common types include **int** (integer), **double** (real number), and **bool** (true/false),

```cpp
/*              C++ Version                */
double areaOfCircle(double r) {
    return M_PI * r * r;
}

int maxOf(int first, int second) {
    if (first > second) {
        return first;
    }
    return second;
}

void printNumber(int n) {
    cout << "I like " << n << endl;
}
```

```python
"""        Python Version          """
def areaOfCircle(r):
    return math.pi * r * r


def maxOf(first, second):
    if first > second:
        return first
    return second



def printNumber(n):
    print("I like " + str(n))
```

```java
/*              Java Ve
private double areaOf
    return M_PI * r *
}

private int maxOf(int
    if (first > secon
        return first;
    }
    return second;
}

private void printNum
    System.out.println("I like " + n);
}
```

```javascript
    console.log("I like " + n);
}
```

If a function does not return a value, its return type should be the cool-but-scary-sounding **void.**

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 (10:30AM 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.
- ***Introduction to Recursion***
  - A new perspective on problem-solving.