Welcome to CS106B!

- Five Handouts Available:
  - Handout 00: Course Information
  - Handout 01: CS106B Calendar
  - Handout 02: Course Placement
  - Handout 03: CS106B and the Honor Code
  - Handout 04: Assignment 0
- These are also all available online on the course website, https://cs106b.stanford.edu.

Who's Here Today?

- Aero/Astro
- Applied Physics
- Bioengineering
- Biology
- Biophysics
- Business
- Cancer Biology
- Chemistry
- Civil/Environmental Engineering
- Communication
- Computer Science
- Creative Writing
- Earth Systems
- East Asian Studies
- Economics
- Education
- Electrical Engineering
- Energy Resources Engineering
- Engineering
- Epidemiology
- Film and Media Studies
- Geophysics
- Global Studies
- Human Biology
- Immunology
- International Policy
- International Relations
- Law
- Management Science
- Materials Science / Engineering
- Math and Computational Science
- Mathematics
- Mechanical Engineering
- Medicine
- Molecular/Cell Physiology
- Music
- Petroleum Engineering
- Physics
- Psychology
- Statistics
- Symbolic Systems
- ***Undeclared!***

Course Staff

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

Head TA: Katherine Erdman
(kerdman@stanford.edu)

The CS106B Section Leaders
The CS106B Course Helpers

Course Website

https://cs106b.stanford.edu

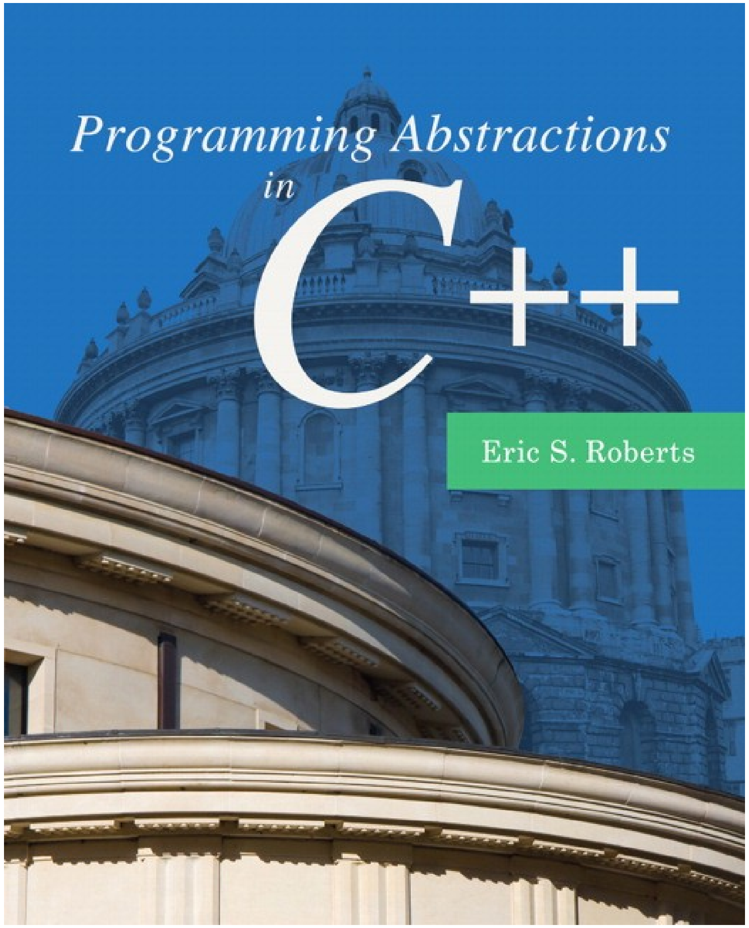
Prerequisites

CS106A

(or equivalent)

(check out our course placement handout if you're unsure!)

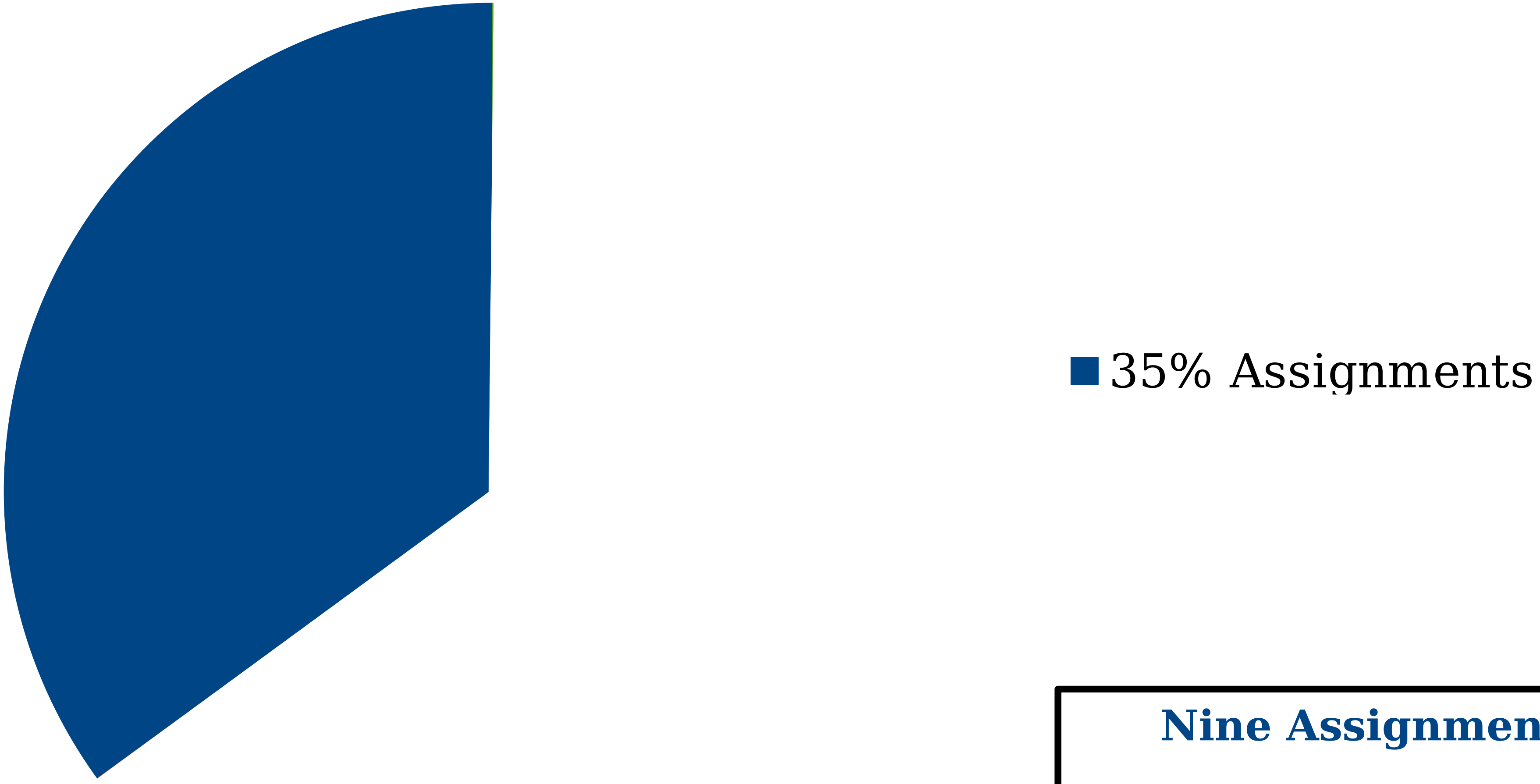
Required Reading



- 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 are many copies available on reserve in the Engineering Library.

Grading Policies

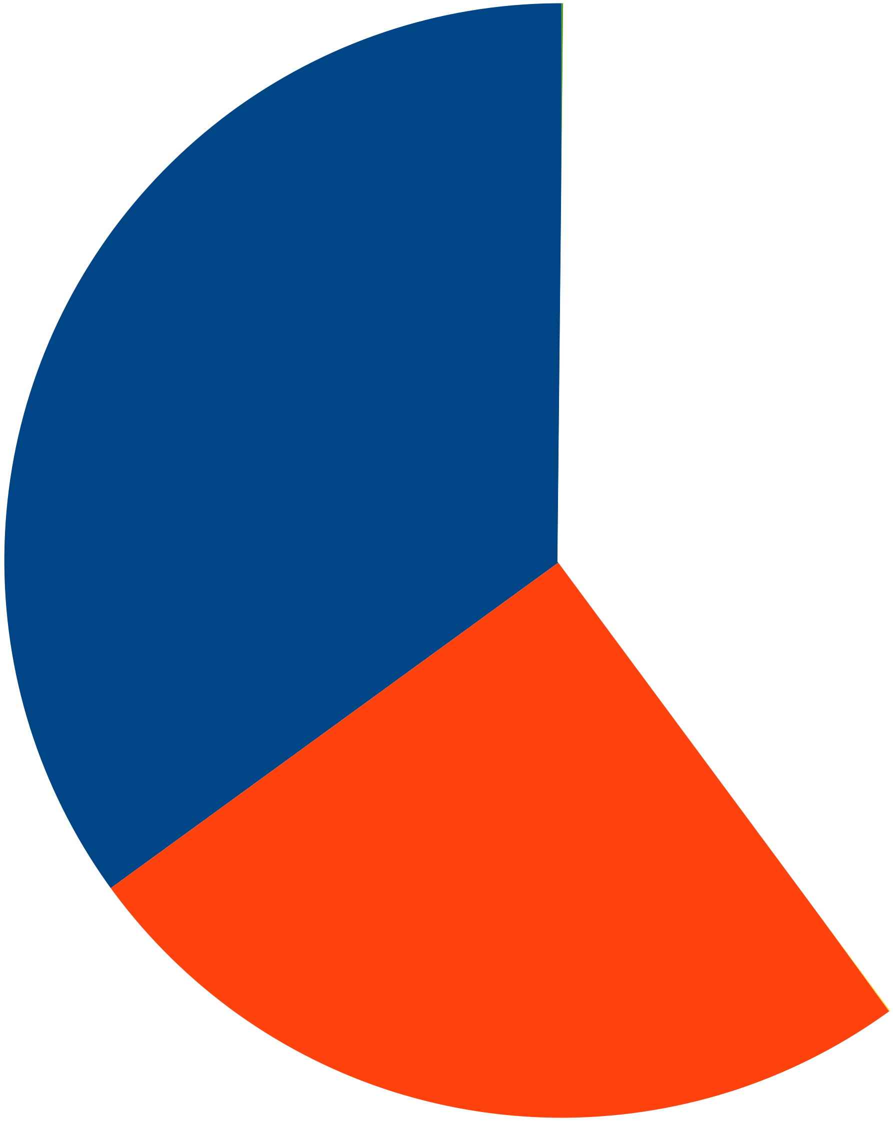
Grading Policies



Nine Assignments

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

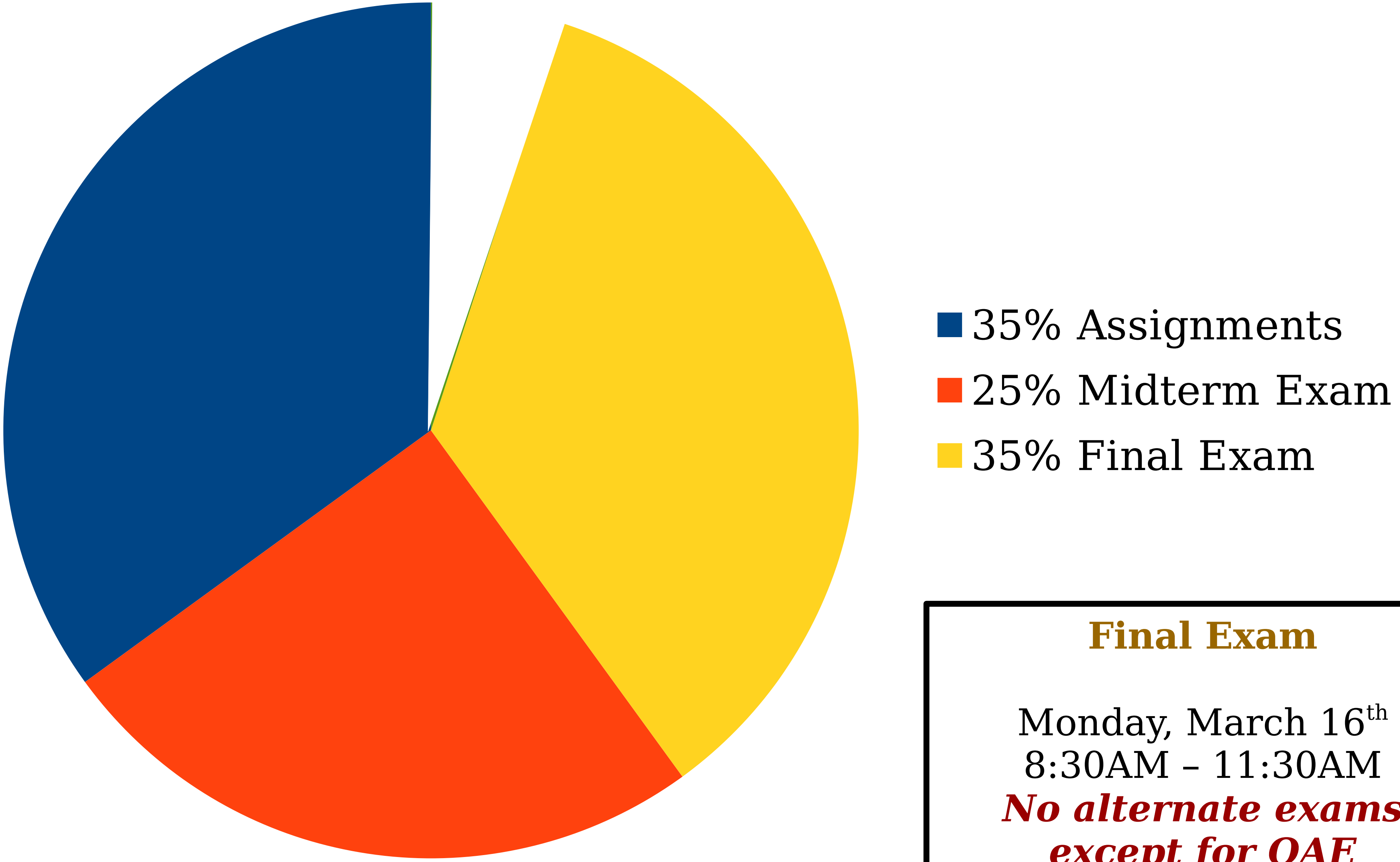
Grading Policies



Midterm Exam

Tuesday, February 11th
7PM – 10PM
Location TBA

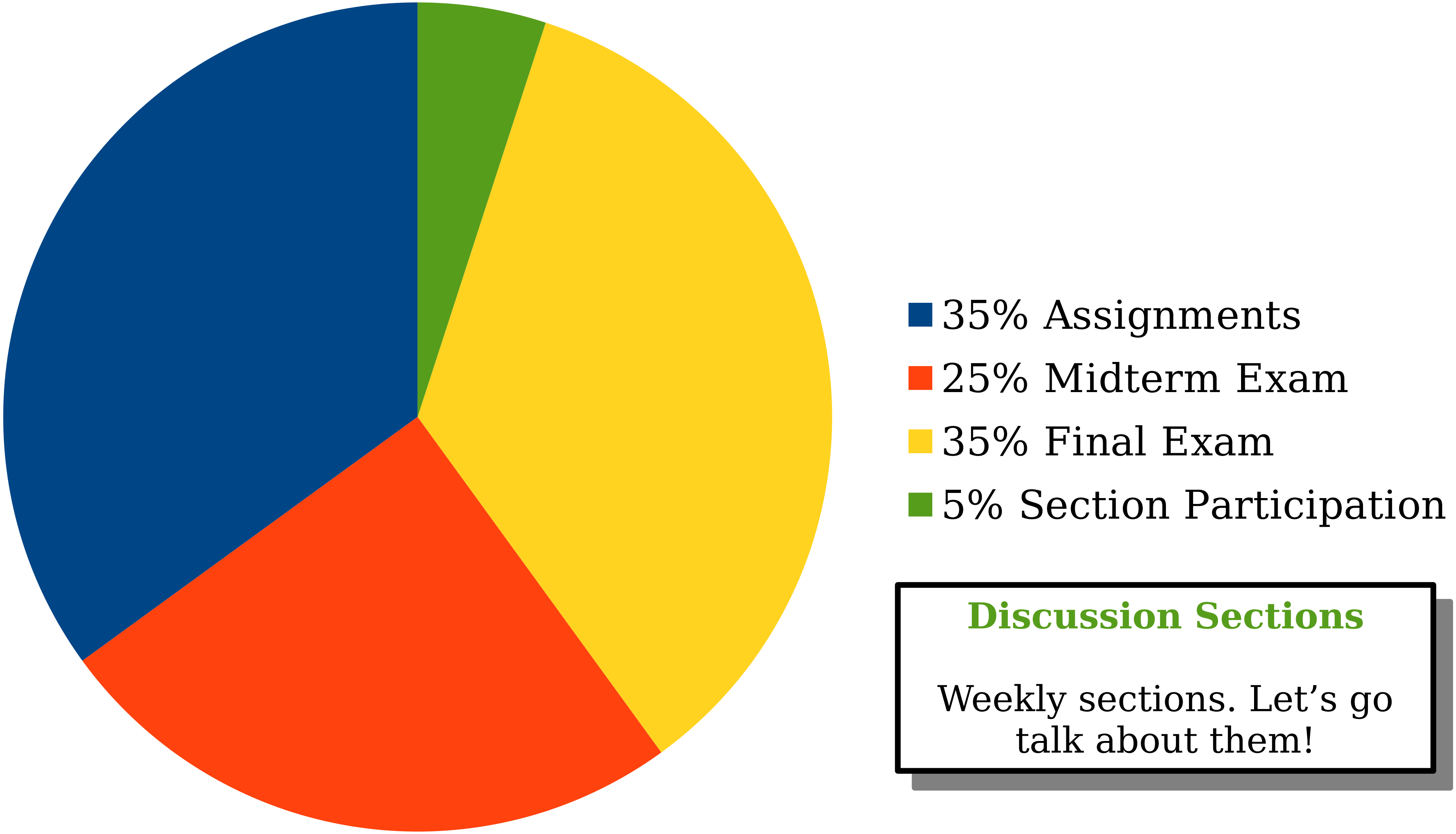
Grading Policies



Final Exam

Monday, March 16th
8:30AM – 11:30AM
No alternate exams except for OAE accommodations.

Grading Policies



Discussion Sections

Weekly sections. Let’s go talk about them!

Discussion Sections

- There are weekly discussion sections in CS106B. Section attendance is required.
- Sign up between Thursday, January 9th at 5:00PM and Sunday, January 12th at 5:00PM by visiting

  http://cs198.stanford.edu/section
- 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.

How Many Units?

```
int numUnits(bool isGrad) {
    if (isGrad) {
        return randomInteger(3, 5); // 3 to 5
    } else {
        return 5;
    }
}
```

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.

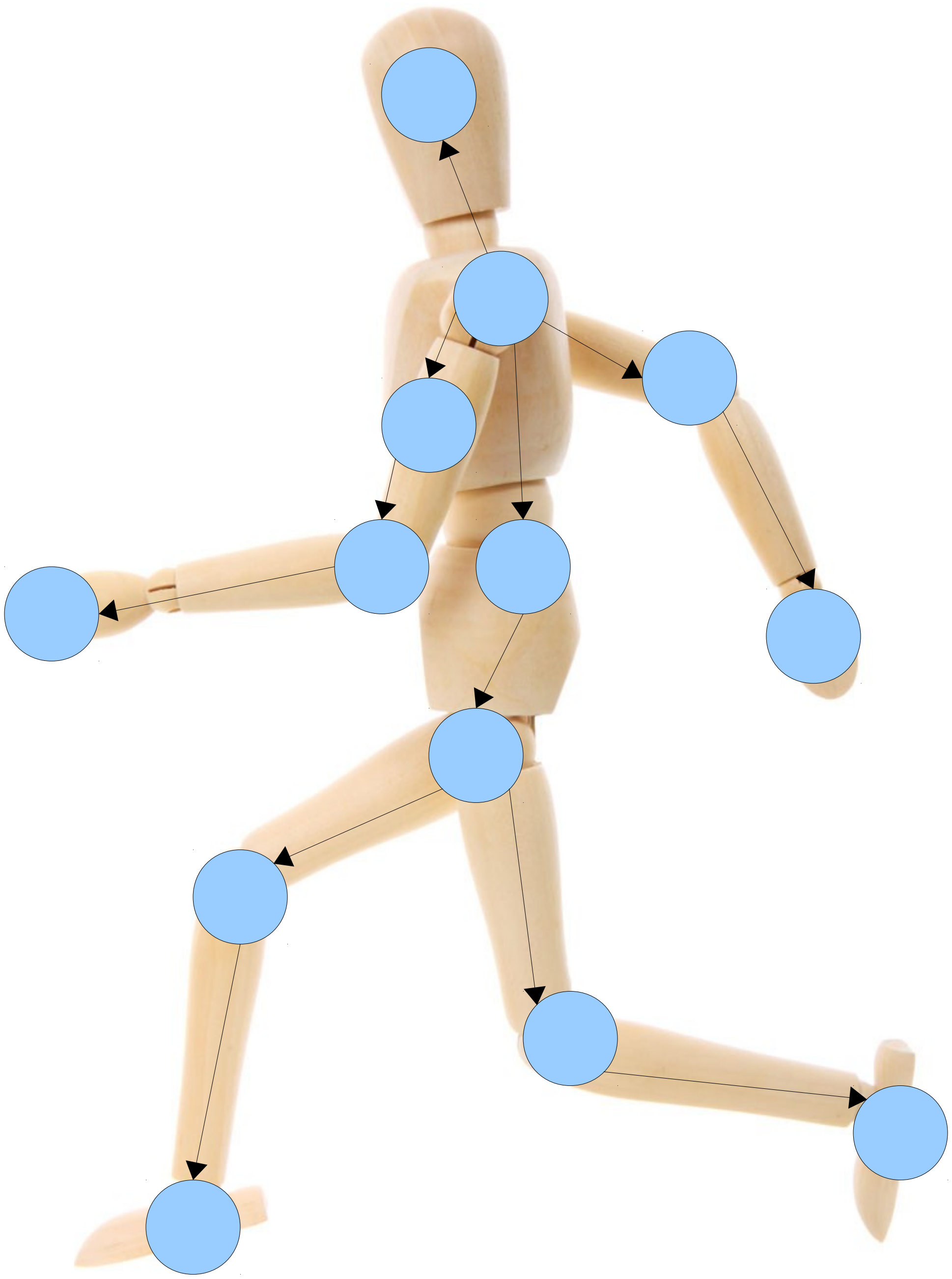
Goals for this Course

Learn how to model and solve complex problems with computers.

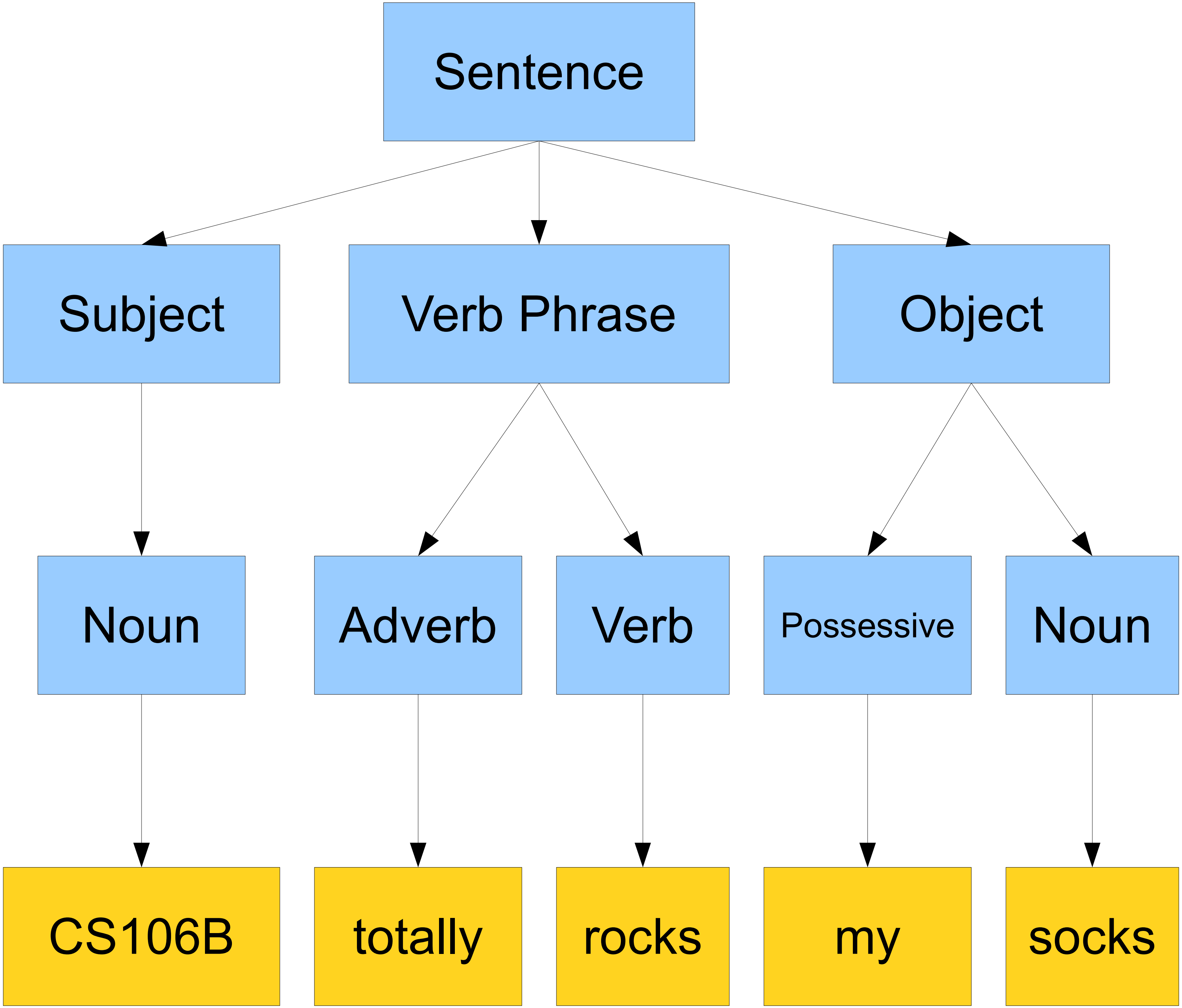
To that end:

- 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

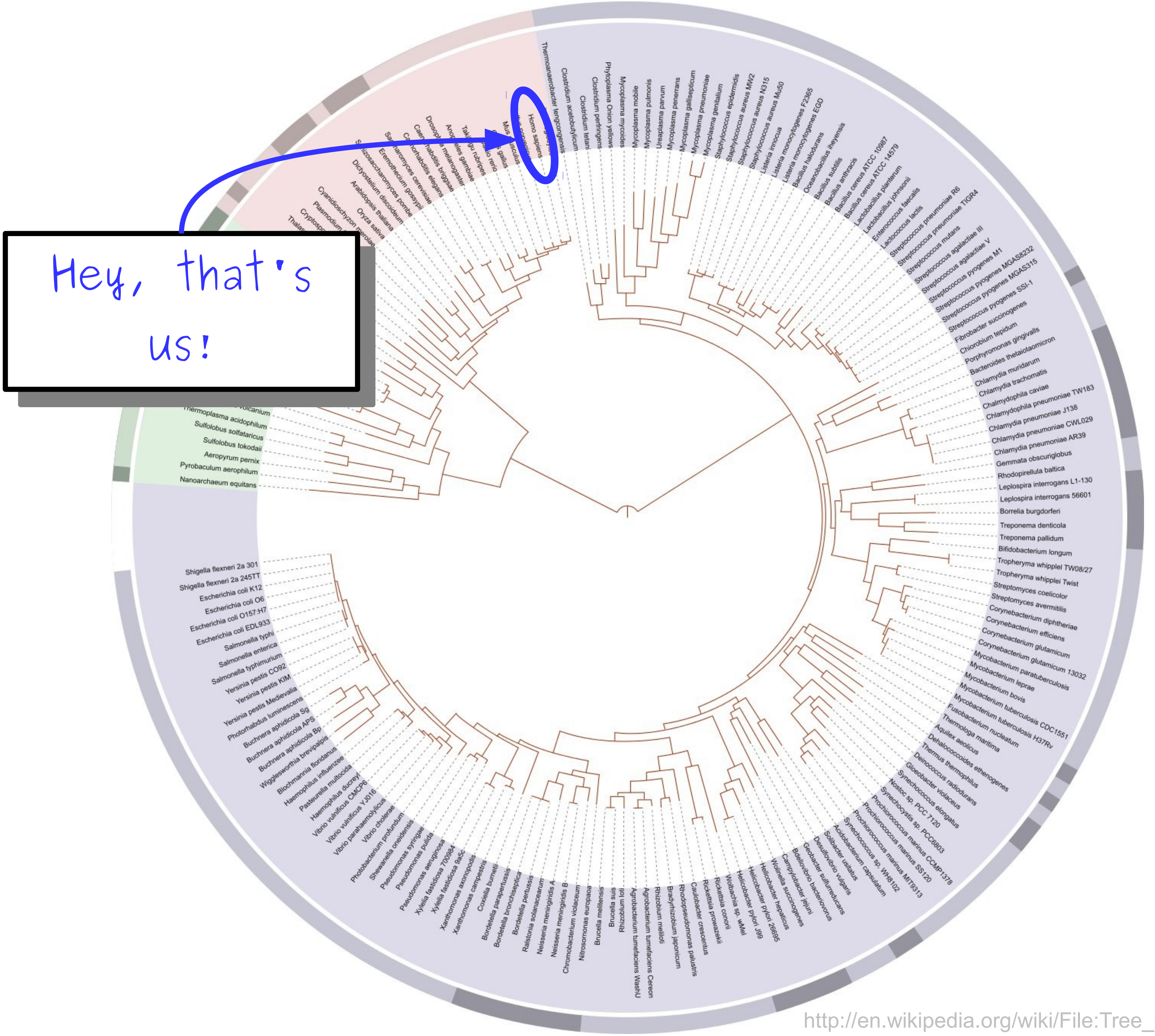
CS106B

totally

rocks

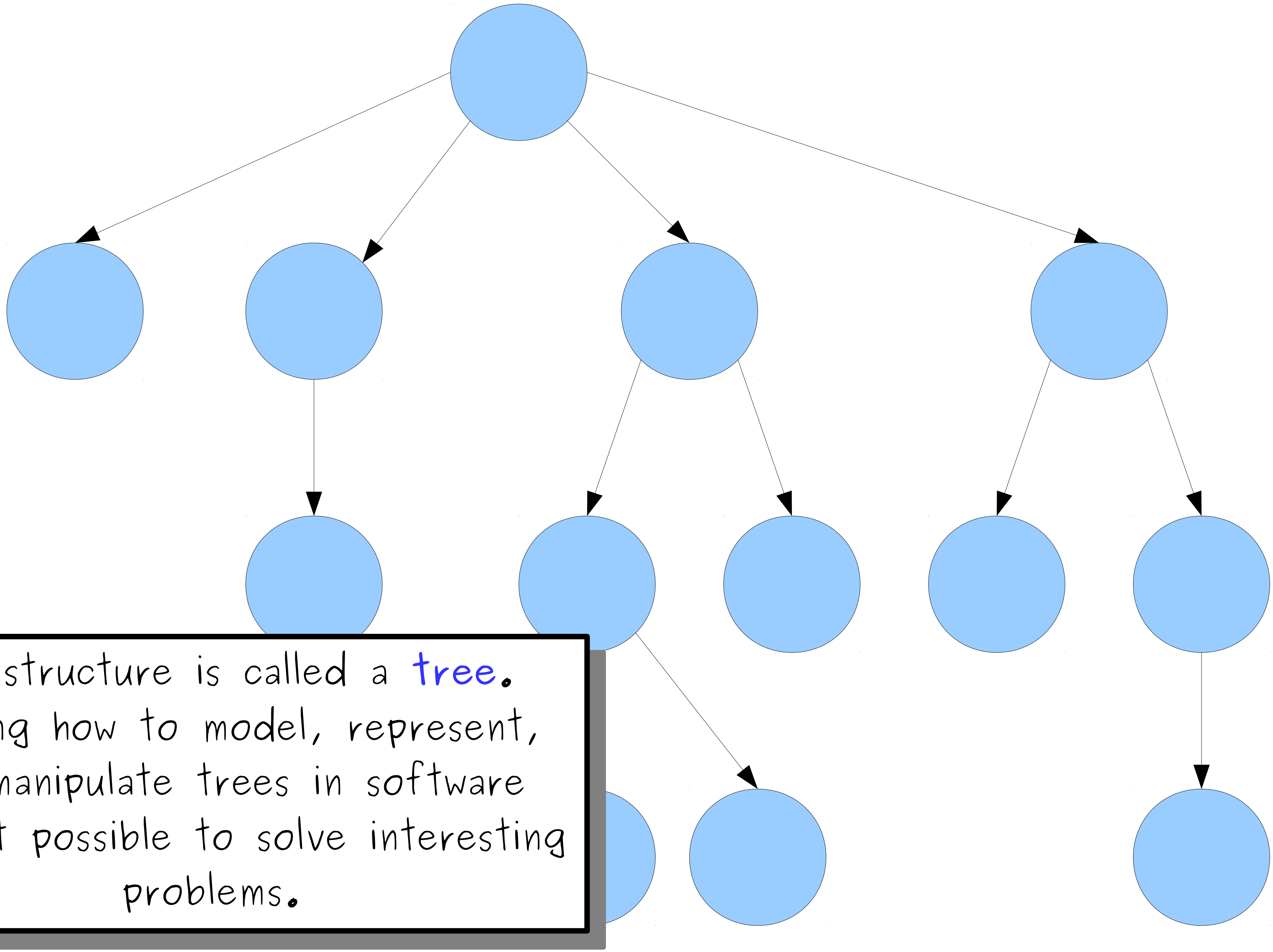
my

socks



Hey, that's us!

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

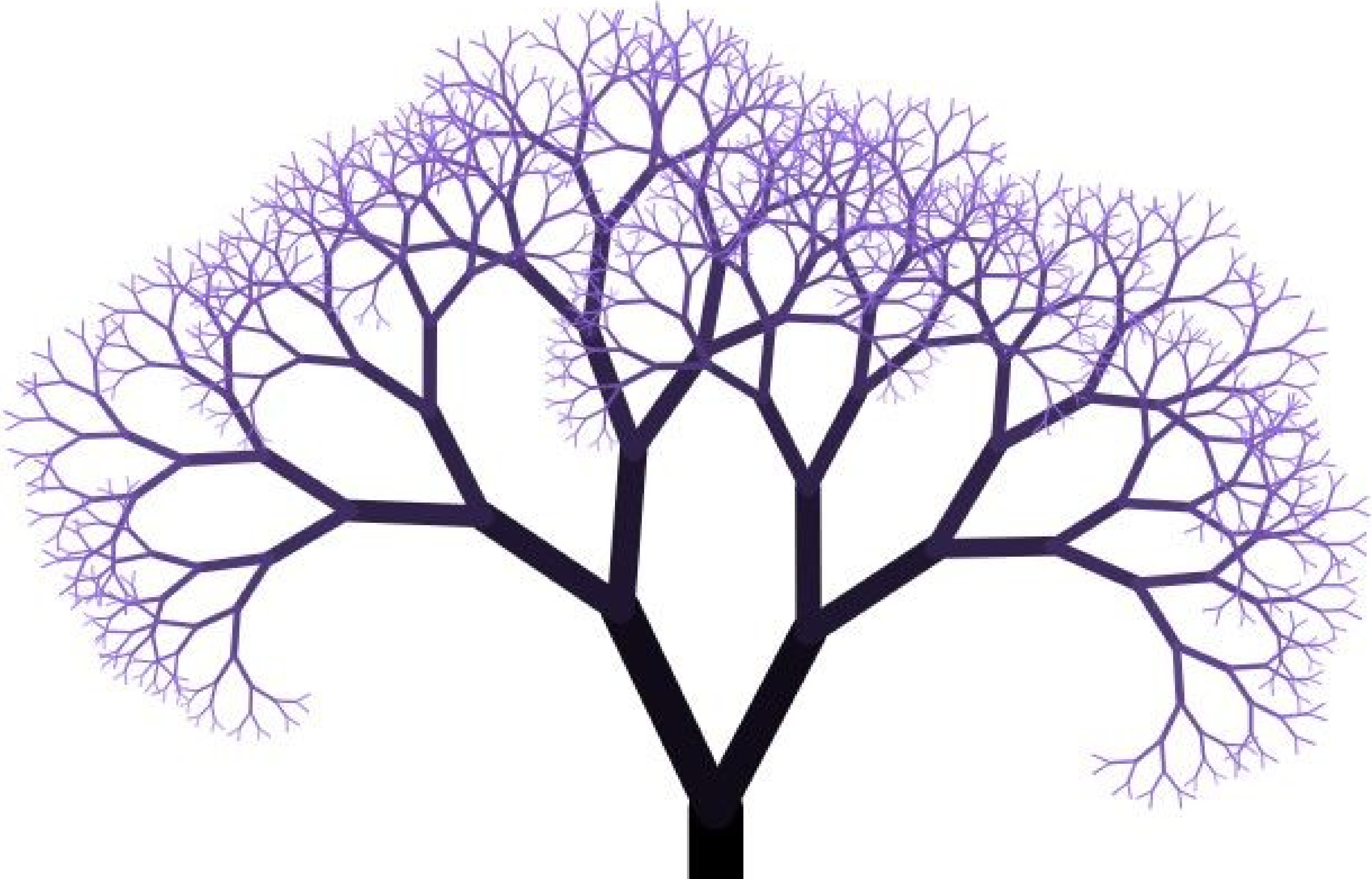
Learn how to model and solve complex problems with computers.

To that end:

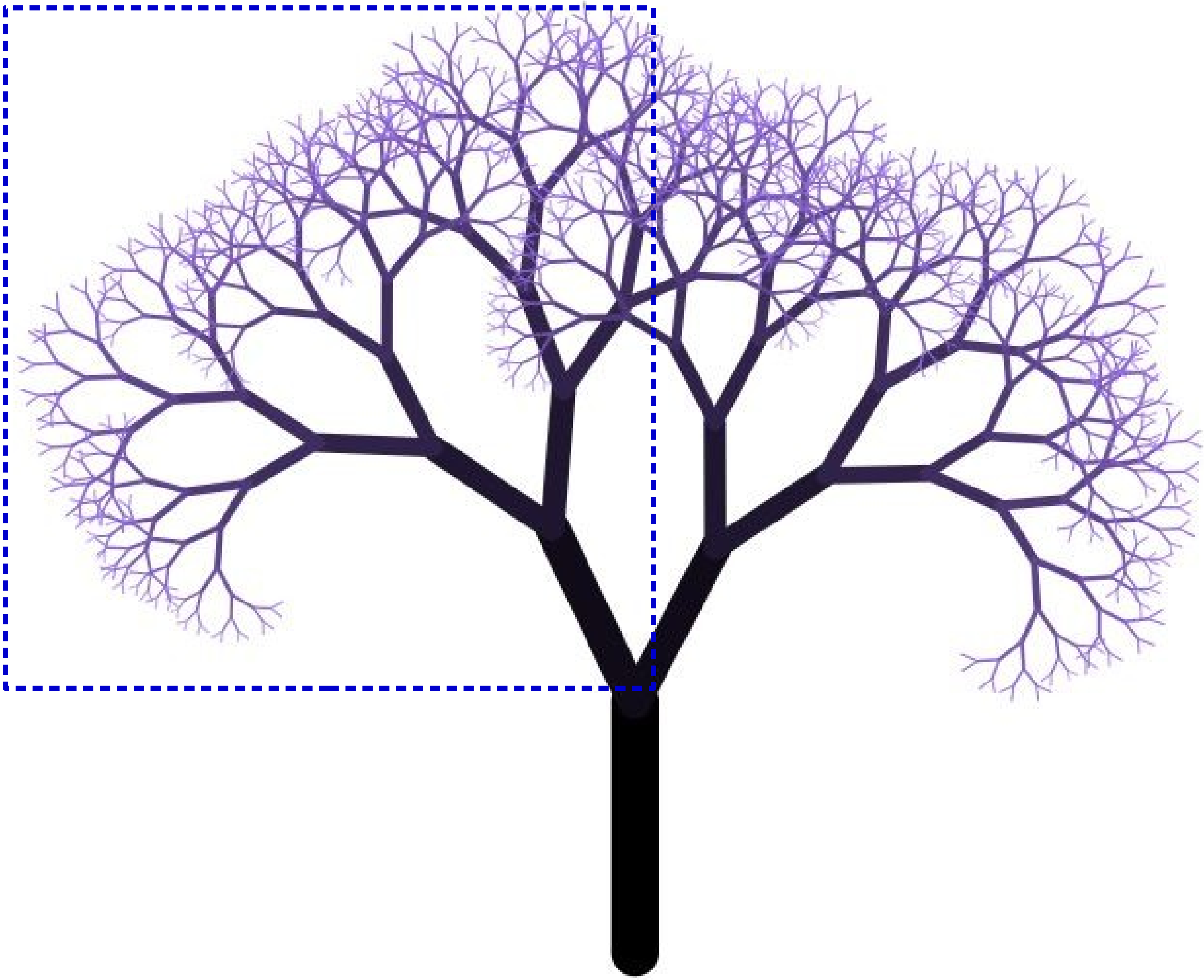
Explore common abstractions for representing problems.

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

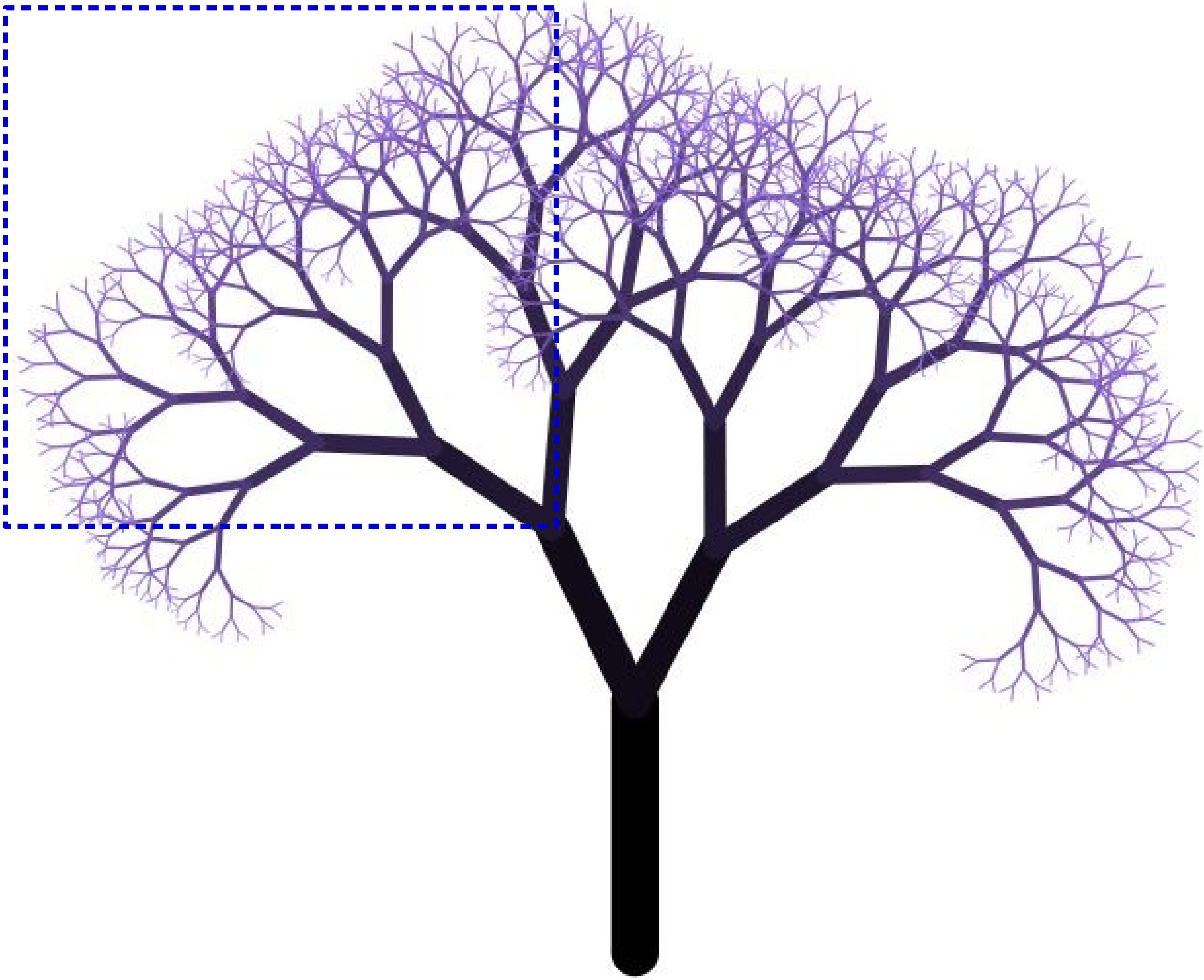
Quantitatively analyze different approaches for solving problems.



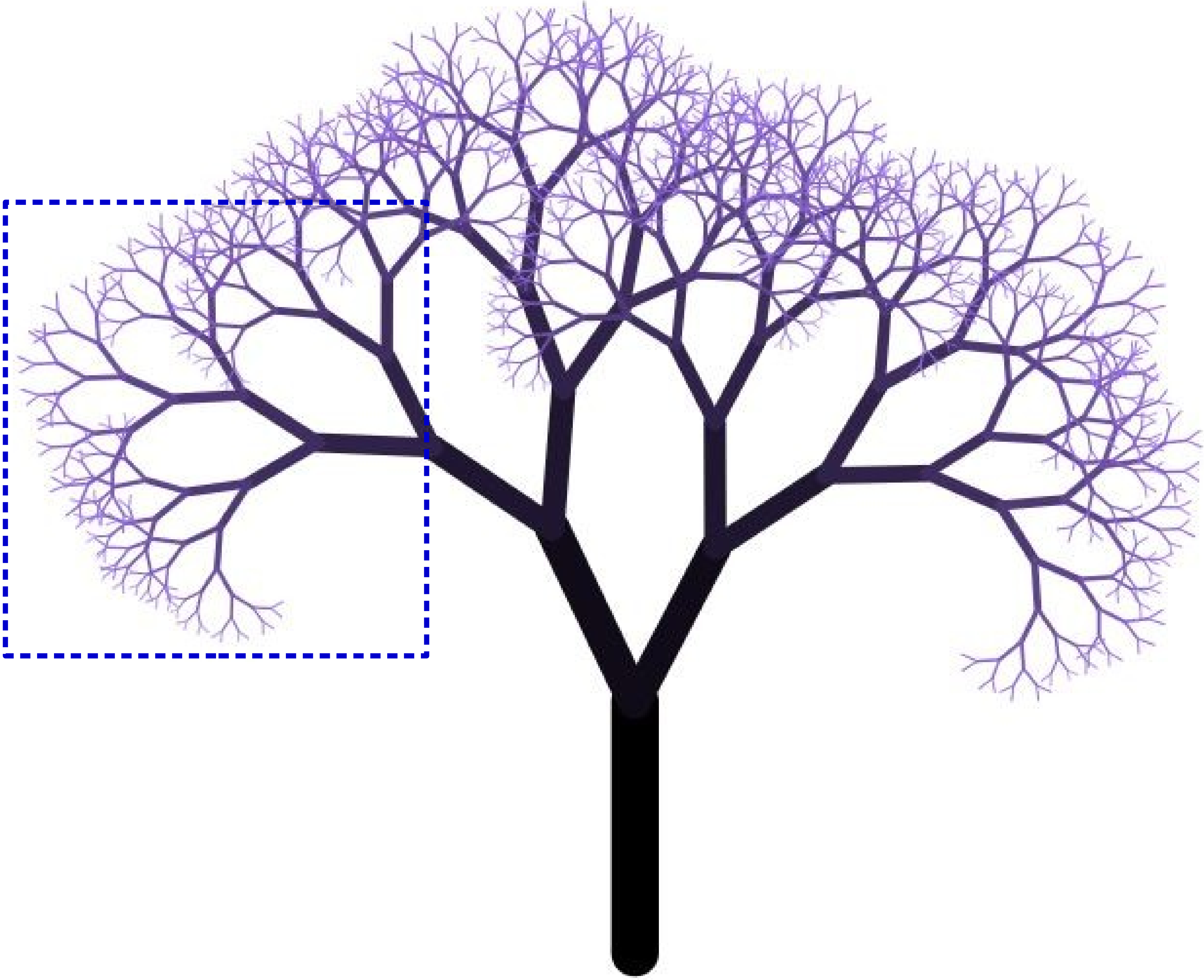
http://www.marketoracle.co.uk/images/2010/Oct/fractal-tree2.jpg



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

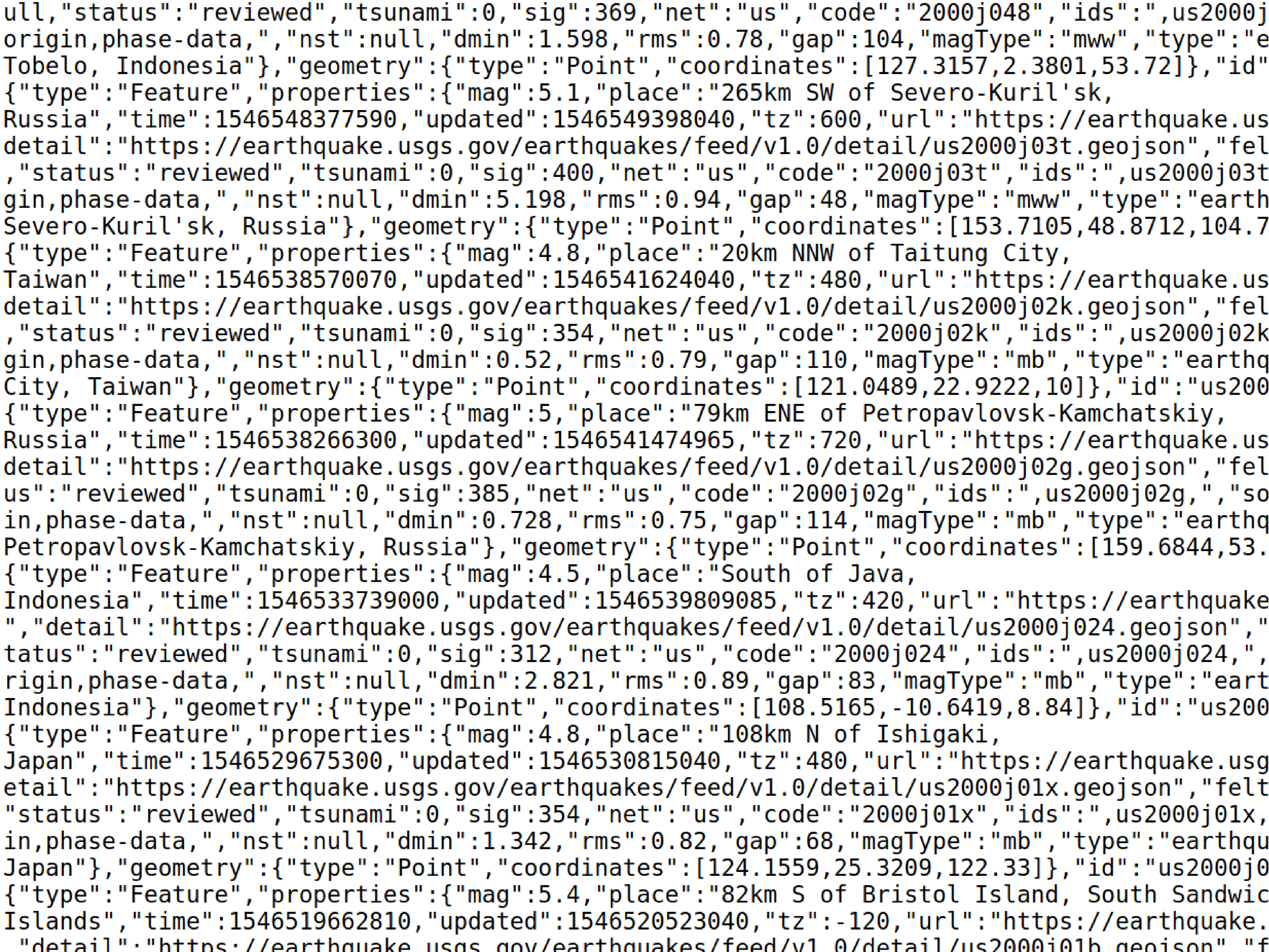
- ***Learn how to model and solve complex problems with computers.***
- To that end:
  - Explore common abstractions for representing problems.
  - Harness recursion and understand how to think about problems recursively.
  - Quantitatively analyze different approaches for solving problems.

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
```

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
- Applied Physics
- Bioengineering
- Biology
- Biophysics
- Business
- Cancer Biology
- Chemistry
- Civil/Environmental Engineering
- Communication
- Computer Science
- Creative Writing
- Earth Systems
- East Asian Studies
- Economics
- Education
- Electrical Engineering
- Energy Resources Engineering
- Engineering
- Epidemiology
- Film and Media Studies
- Geophysics
- Global Studies
- Human Biology
- Immunology
- International Policy
- International Relations
- Law
- Management Science
- Materials Science / Engineering
- Math and Computational Science
- Mathematics
- Mechanical Engineering
- Medicine
- Molecular/Cell Physiology
- Music
- Petroleum Engineering
- Physics
- Psychology
- Statistics
- Symbolic Systems
- ***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 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.

```python
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 ofits 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;
}
```

```
#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.

```
#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.

```
#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!)

```
#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.

```
#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.

```
#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.

Functions in C++

C++ Functions

- Functions in C++ are similar to methods in Java and functions in JavaScript / Python:
  - They're pieces of code that perform tasks.
  - They (optionally) take parameters.
  - They (optionally) return a value.
- Here's some functions:

```
double areaOfCircle(double r) {
    return M_PI * r * r;
}
```

If a function returns a value, the type of the returned value goes here. (**double** represents a real number.)

```
void printBiggerOf(int a, int b) {
    if (a > b) {
        cout << a << endl;
    } else {
        cout << b << endl;
    }
}
```

If a function doesn't return a value, put the word **void** here.

The `main` Function

- A C++ program begins execution in a function called `main` with the following signature:

```
int main() {
    /* … code to execute … */
    return 0;
}
```

- By convention, `main` should return 0 unless the program encounters an error.

The function `main` returns an integer. Curious where that integer goes? Come talk to me after class!

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 at the start of class (11:30AM). Starter files and assignment handout 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.
  - Reference parameters.
  - Introduction to recursion.