CS 4110
Programming Languages & Logics

Lecture 1
Course Overview

JavaScript

```
[] + []
{} + []
[] + {}
{} + {}
```

From *Wat:* `https://www.destroyallsoftware.com/talks/wat`

Java

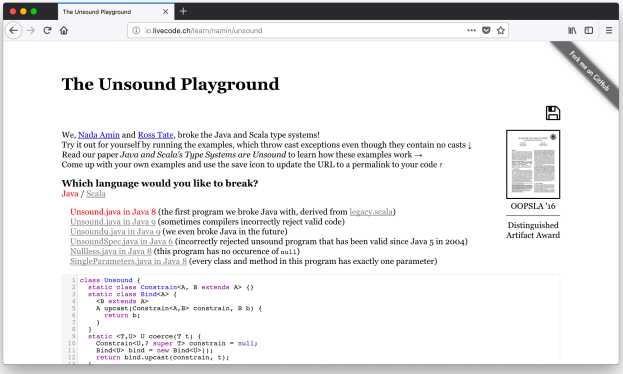
```
class A {
    static int a = B.b + 1;
}

class B {
    static int b = A.a + 1;
}
```

Python

```
a = [1], 2
a[0] += [3]
```

Java and Scala



Nada Amin and Ross Tate: `http://io.livecode.ch/learn/namin/unsound`

Design Desiderata

Question: What makes a good programming language?

Design Desiderata

Question: What makes a good programming language?

One answer: A good language is one people use.

Design Desiderata

Question: What makes a good programming language?

One answer: A good language is one people use.

Wrong! Is JavaScript bad? What's the best language?

Design Desiderata

Question: What makes a good programming language?

One answer: A good language is one people use.

Wrong! Is JavaScript bad? What's the best language?

Some good features:

- Simplicity (clean, orthogonal constructs)
- Readability (elegant and intuitive syntax)
- Safety (guarantees that programs won't "go wrong")
- Modularity (support for composition and collaboration)
- Efficiency (it's possible to write a good compiler)

Design Challenges

Unfortunately these goals almost always conflict.

- Types provide strong guarantees but restrict expressiveness.
- Runtime checks eliminate errors but hurt performance.
- A language that's good for quick prototyping might not be the best for long-term development.

Design Challenges

Unfortunately these goals almost always conflict.

- Types provide strong guarantees but restrict expressiveness.
- Runtime checks eliminate errors but hurt performance.
- A language that's good for quick prototyping might not be the best for long-term development.

A lot of research in programming languages is about discovering ways to gain without adding (too much) pain.

Formal Semantics

Central theme: what do programs mean, precisely?

Three Classic Approaches

- Operational
  - Models program by its execution on abstract machine
  - Useful for implementing compilers and interpreters

Formal Semantics

Central theme: what do programs mean, precisely?

Three Classic Approaches

- Operational
  - Models program by its execution on abstract machine
  - Useful for implementing compilers and interpreters
- Axiomatic
  - Models program by the logical formulas it obeys
  - Useful for proving program correctness

Formal Semantics

Central theme: what do programs mean, precisely?

Three Classic Approaches

- Operational
  - Models program by its execution on abstract machine
  - Useful for implementing compilers and interpreters
- Axiomatic
  - Models program by the logical formulas it obeys
  - Useful for proving program correctness
- Denotational
  - Models program literally as mathematical objects
  - Useful for theoretical foundations

Formal Semantics

Central theme: what do programs mean, precisely?

Three Classic Approaches

- Operational
  - Models program by its execution on abstract machine
  - Useful for implementing compilers and interpreters
- Axiomatic
  - Models program by the logical formulas it obeys
  - Useful for proving program correctness
- Denotational
  - Models program literally as mathematical objects
  - Useful for theoretical foundations

Question: Few languages have a formal semantics. Why?

Formal Semantics

Too Hard?

- Real languages are complex
- Notation can gets very dense
- Sometimes requires developing new mathematics
- Not (yet?) cost-effective for everyday use

Overly General?

- Explains the behavior of a program on *every* input
- Most programmers are content knowing the behavior of their program on *this* input (or these inputs)

Okay, so who needs semantics?

Who Needs Semantics?

Unambiguous Description

- Anyone who wants to design a new language feature
- Understand interactions with other features
- *The* standard tool in PL research

Exhaustive Reasoning

- When you need to consider all inputs—e.g., critical software
- Compilers and interpreters
- Analysis and verification tools

Program Synthesis and LLMs

- Constrain code generation using syntax *and* semantics
- Prove theorems that increase trust in generated code

Course Staff

Instructor

Nate Foster

Teaching Assistants

Annabel Baniak
Hanxi Chen
Serena Duncan
Jeffrey Huang
Stephanie Ma
Luis Hernandez Rocha

Prerequisites

Mathematical Maturity

- Much of this class will involve formal reasoning
- Set theory, formal proofs, induction

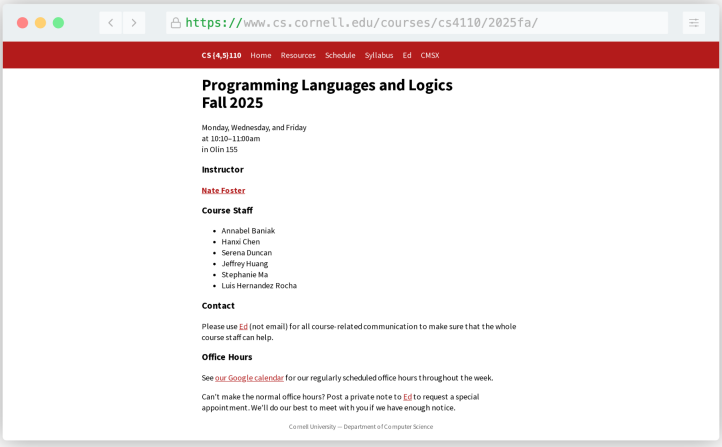
Programming Experience

- Comfortable using a functional language
- For Cornell undergrads: CS 3110 or equivalent

Interest (having fun is a goal!)

If you don't meet these prerequisites, please get in touch.

Course Website



http://www.cs.cornell.edu/courses/cs4110/2025fa/

Course Work

Homework (20% of final grade)

- 10 assignments
  - One per week
  - Except for weeks with prelims and breaks
- Can work with *at most one* partner
- Usually due on Thursday night at 11:59pm
- Automatic 24-hour extension without penalty
- Score capped at 85%
- Lowest score dropped

Course Work

In-class Preliminary Exams (40% of final grade)

- October 10 (Friday)
- November 10 (Monday)

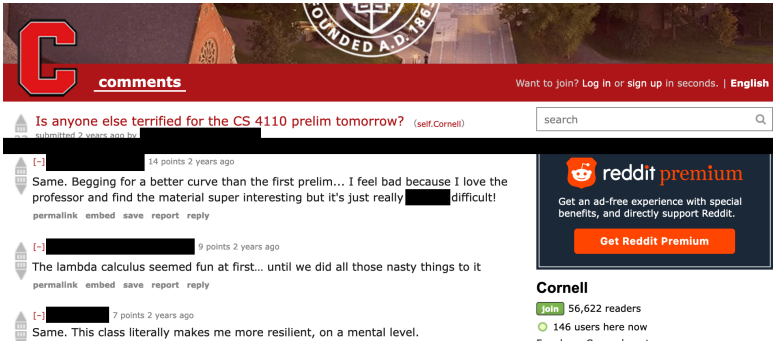
Final Exam (35% of final grade)

- Date TBD (by registrar)

Participation (5% of your grade)

- Introduction survey (on CMSX now!)
- Mid-semester feedback
- Course evaluation

The Difficulty You Can Expect



CS 4110 vs. CS 5110

The difference is:

- CS 4110 is for undergrads (exclusively);
- CS 5110 is for grad students (exclusively).

Everything else is the same, except that CS 5110 students do an extra “expanded version” of a solution after each exam.

Academic Integrity

Some simple requests:

1. You are here as members of an academic community. Conduct yourself with integrity.

2. Problem sets must be completed with your partner, and only your partner. You must *not* consult other students, alums, friends, Google, GitHub, StackExchange, Course Hero, etc.!

3. If you aren't sure what is allowed and what isn't, please ask.

Generative AI

We'll use a "choose your own adventure" policy

1. Don't generative AI (disable Copilot, Cursor, etc.)

2. Freely use generative AI, but write a brief disclosure.

Respect in Class

I will hold all communication (in class & online) to a high standard for inclusiveness. It may not target anyone for harassment, and it may not exclude specific groups.

Examples:

- Don’t talk over other people.
- Don’t use gendered pronouns to refer to all people.
- Avoid language that has a good chance of seeming inappropriate to others.

If anything doesn’t meet these standards, contact the instructor.

Accommodations and Wellness

- I will provide reasonable accommodations for religious observences and for students with documented disabilities (e.g., physical, learning, psychiatric, vision, hearing, etc.).
- If you already know you will need accommodations, please let me know within the first three weeks of the semester.
- We will use Cornell's Alternative Testing Program for all exam accommodations.
- If you are experiencing undue personal or academic stress at any time during the semester (or if you notice that a fellow student is), contact me, Engineering/A&S Advising, or Gannett.

What's Next?

- Introductory Survey: Fill out on CMSX by Thursday
- Foster Office Hours: Wednesday 2:30-3:30, or by appointment