CST 372: Programming Languages

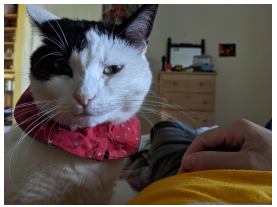
Introduction, Racket, and your first interpreter

Introductions: Me

Sam, Prof Stites, Prof Sam, Dr Stites, Dr Sam, …

- New instructor! Be forgiving
- Seven years industry (start-up to super-large, SE-to-ML, 100+ interviews)
- Research: Programming Languages, Uncertainty, Software Engineering
- Open source contributor &
  Google Summer of Code mentor (in an obscure language)







Introductions: You and this class

Card with your name:

- Your motivation for taking this class / what you think this class is about
- The programming language you are most curious about (/ like)
- What *is* a programming language?

Unnamed(?) card:

- “AI is the best thing since sliced bread”
  - agree? disagree? it’s complicated?

Today

- Pt. 1: Introduction
- Pt. 1: Course structure and logistics
- <5min break>
- Pt. 2: (Live coding) Intro to Plait & CalcLang
- <5-10min break, depending on time>
- Pt. 3: (Live coding) Interpreters, Interpreters, interpreters

I highly encourage everyone to install Racket during this first part of the class! https://racket-lang.org

- Before next class: syllabus and course questionnaire

What is a “programming language”?

“A programming language is a system of notation for writing computer programs.”



https://en.wikipedia.org/wiki/Programming_language

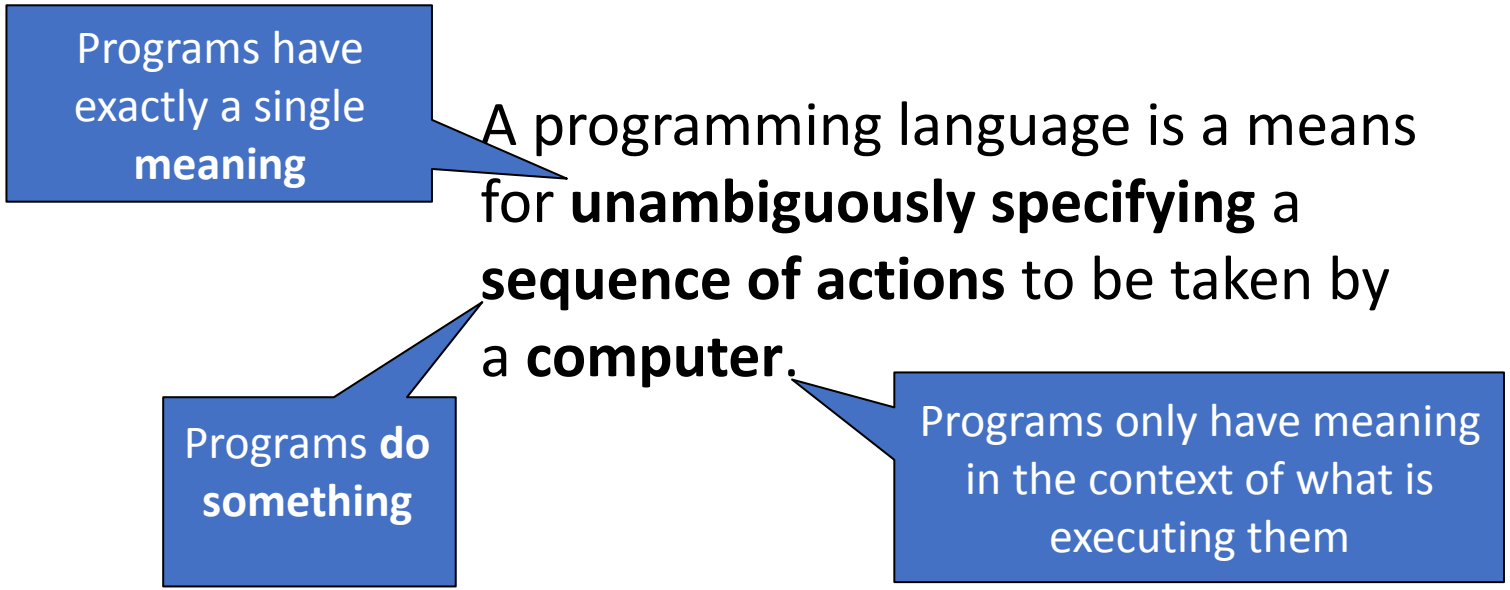
What is a “programming language”?

“Computer programming language, any of various languages for expressing a set of detailed instructions for a digital computer.”



https://www.britannica.com/technology/computer-programming-language

What is a “programming language”?



Consequences of Programming Language Design

Have you ever debugged a program that did something completely absurd?

A crazy bug in class

/ stupid error in a side project

Consequences of Programming Language Design

C++

```
#include <iostream>

int main() {
   while(true) {}
}
void go_home_c_plus_plus_you_are_drunk() {
    std::cout << "lol" << std::endl;
}
```

Compiled in Godbolt

Consequences of Programming Language Design

What’s the point?

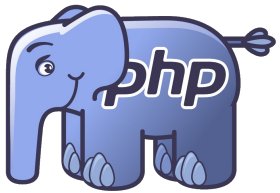
Consequences of Programming Language Design

Java

```
public class BillionDollarMistake {
    public static void main(String[] args) {
        Object obj = null;
        obj.hashCode();
    }
}
```

Consequences of Programming Language Design

Moral of this story



😭😭😭😭

Consequences of Programming Language Design

Moral of this story



Memory Safe Languages: Reducing Vulnerabilities in Modern Software Development

Executive summary

Memory safe languages (MSLs) are gaining momentum. In 2022, the National Security Agency (NSA) released a cybersecurity information sheet (CSI), "Software Memory Safety." [1] In 2023, the Cybersecurity and Infrastructure Security Agency (CISA) published the joint guide, "The Case for Memory Safe Roadmaps," [2] and in 2024, the White House issued "Back to the Building Blocks: A Path Toward Secure and

Languages have Two Parts

Languages have Two Parts

Syntax

(Form)



What does it look like?

Semantics

(Function)

“Your debt is canceled”

What does it do?

Programming Languages have Two Parts

Syntax

Python

```
x = 5
print(x)
```

What does it look like?

Semantics

- Create a local variable called “x” and assign it equal to 5
- Print the value of “x” to the console

What does it do?

Is this *really* what the program does? What level of detail do we want?

Programming Languages have Two Parts

Syntax

Javascript

```
let x = 5;
console.log(x)
```

What does it look like?

Semantics

- Create a local variable called “x” and assign it equal to 5
- Print the value of “x” to the console

What does it do?

Programming Languages have Two Parts

Syntax

Racket

```
(define x 5)
(display x)
```

What does it look like?

Semantics

- Create a local variable called “x” and assign it equal to 5
- Print the value of “x” to the console

What does it do?

This course is all about precisely defining programming languages

Syntax

Formal descriptions as *grammars*

Semantics

Programs that run programs

Interpreters!

- Grow big languages out of small ones
- Implement new languages

Duality of Programming Languages:

Programming languages are *mathematical constructs*

$$\frac{x = 5 \qquad y = 10}{x + y \Downarrow 15}$$

Duality of Programming Languages:

Programming languages are *social constructs*





Questions you may have?

Why are there so many programming languages?

Which language should I learn? Should I use?

Are some languages worse than others? Better? How can I compare them?

What distinguishes one language from another?

Why are new languages being made today?

How are new languages made?

Why study programming languages?

- Be a more effective programmer
  - Learn how to **choose languages** for your problem
  - Learn how to **design new languages** if needed
- Become equipped to learn new languages quickly
- Be prepared for an evolving world
- Enjoy an aesthetic journey through this elegant field (subjective)

Course structure and logistics

Syllabus and Course Website

- Hw Submissions: Blackboard
- Q+A & Announcements: Discord
- (optional) Pairing: Tuple.app (invite)
- Course website: https://dev.stites.io/cst372
  - Assignments
  - Deadlines
  - Syllabus
  - Course notes (+ these slides)

You are expected to read and be aware of all content in the course syllabus!

Module 1: Introduction

- Become familiar with *inductively defined data* and *functional programming in Racket*

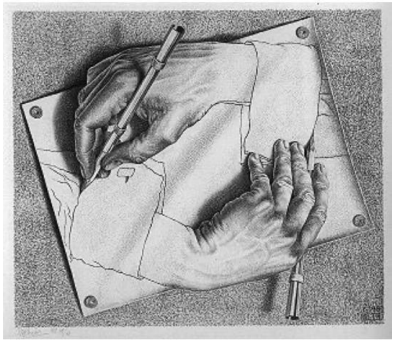
- See the design recipe for programming with inductively defined recursive data

- Become comfortable with core functional programming idioms
  - First-class functions
  - Lists
  - Pattern matching

Module 2: The Lambda Calculus

- We will implement a tiny programming language one feature at a time

1. Calculator-lang
2. Let-lang (binding, substitution)
3. Lambda calculus (functions)
4. Programming in the lambda calculus
5. Implementing the lambda calculus e
6. Recursion in the lambda calculus



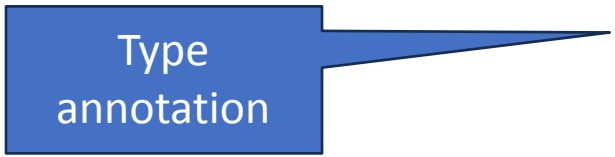
Schedule is subject to change!

Module 2: Why grow the lambda calculus?

- See how a real expressive programming language is made from start to finish
- Learn how to understand formal mathematical descriptions of programs and translate those into implementation
- Learn to appreciate the value of expressive language features by programming in a very restricted language without any features.

Module 3: Types

- Types are many things
  - A form of *checked specification* for programs
  - A way of *constraining the expressivity* of programs
- Example: Java




```
import java.util.Scanner;

public class HelloWorld {

    public static void main(String[] args) {

        // Creates a reader instance which takes
        // input from standard input - keyboard
        Scanner reader = new Scanner(System.in);
        System.out.print("Enter a number: ");

        // nextInt() reads the next integer from the keyboard
        int number = reader.nextInt();

        // println() prints the following line to the output
        System.out.println("You entered: " + number);
    }
}
```

Module 3: Types

Types can’t prevent all bugs! Recall:

```
public class BillionDollarMistake {
    public static void main(String[] args) {
        Object obj = null;
        obj.hashCode();
    }
}
```

Module 3: Types



FEBRUARY 26, 2024

Press Release: Future Software Should Be Memory Safe

ONCD › BRIEFING ROOM › PRESS RELEASE

Leaders in Industry Support White House Call to Address Root Cause of Many of the Worst Cyber Attacks

Read the full report here

WASHINGTON – Today, the White House Office of the National Cyber Director (ONCD) released a report calling on the technical community to proactively reduce the attack surface in cyberspace. ONCD makes the case that technology manufacturers can prevent entire classes of vulnerabilities from entering the digital ecosystem by adopting memory safe programming languages. ONCD is also encouraging the research community to address the problem of software measurability to enable the development of better diagnostics that measure cybersecurity quality.

https://www.whitehouse.gov/oncd/briefing-room/2024/02/26/press-release-technical-report/

Module 3: Why study types?

- You will encounter type systems in your day-to-day programming
- Type systems are getting increasingly sophisticated (memory safety, Rust)
- Gaining a good appreciation of how type systems are designed and implemented prepares you for a more well-typed future

Module 4: Control

- Programs don’t run in order: control flow can jump around
  - If expressions
  - Function calls
  - Exceptions
  - Async/await
  - Callbacks
- How do we add these features to our languages? How are they implemented?

Module 4: Why study control?

- Learn how to implement your own control-flow primitives
  - Lets you add them to languages that don't have them!
  - More deeply understand how compilers are implemented
  - Understand the cost of using various control-flow operations

Module 5: Abstract Machines

- How do programs *actually run* on a simple model of a CPU?

- We will explore memory safety, performance, and dynamic safety.

Module 5: Why study Abstract Machines?

- This will give you a powerful model of how programs are evaluated on low-level hardware

- Abstract machines are *abstract* and works both at the lowest-level of execution, but also is compatible with high-level reasoning like type systems (or static analysis).

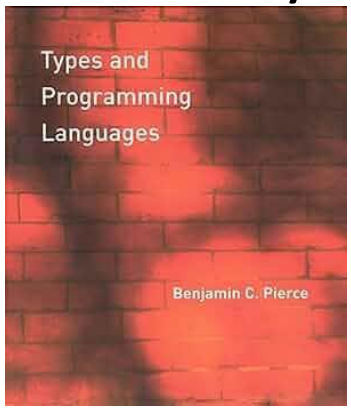
Module 5: Proof Assistants

- This model will explore how a program can be used to prove properties of and reason about programming languages.

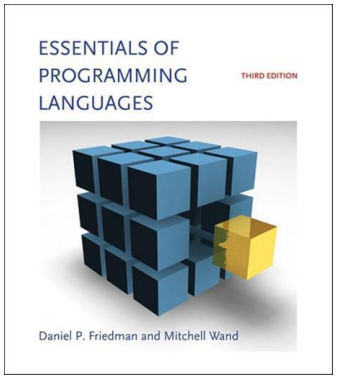
- We will review some basic properties we have already seen and formally prove these statements, using code.

Course resources

- Course notes posted on the main course webpage
- Supplementary textbooks:



Types and Programming Languages
Pierce



Essentials of Programming Languages
Friedman and Wand

Questions?

Introduction to Racket