

S

**CIS352 –
Principles of Programming Languages
Spring 2025**

**Instructor: Kris Micinski
TA: Neda Abdolrahimi**

Course Website:

<https://kmicinski.com/cis352-s25>

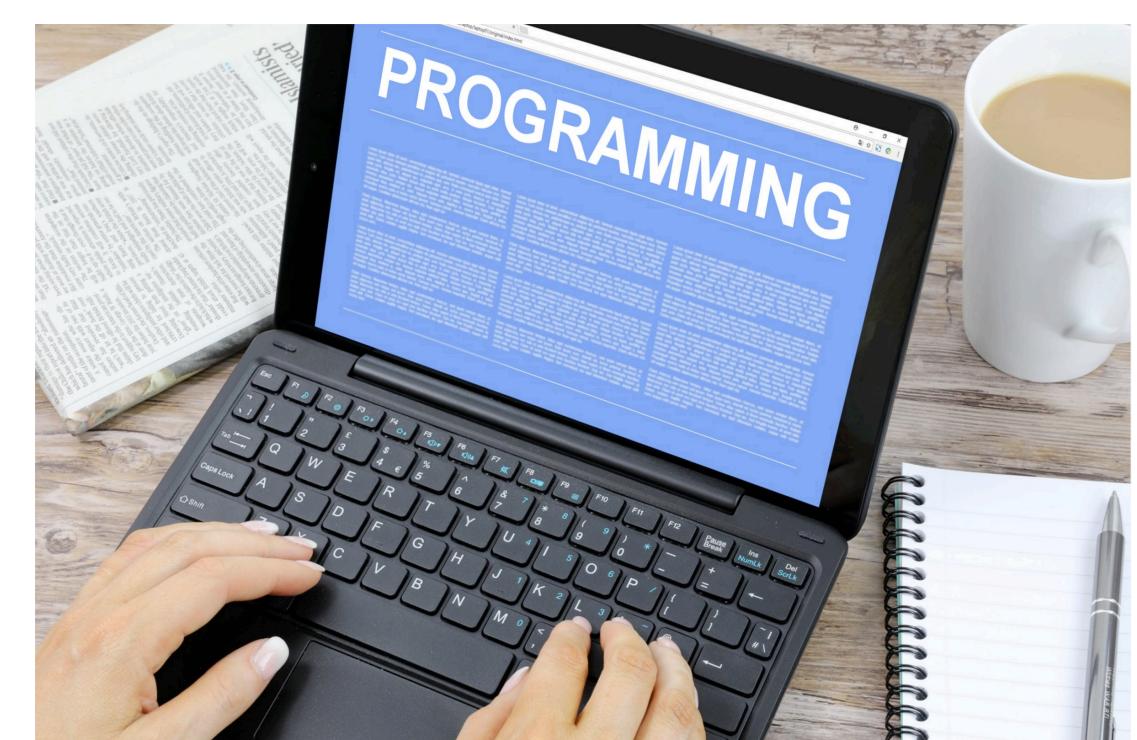




We use writing to help ourselves structure our thoughts—
revising, editing, restarting along the way

This class examines the process of writing and understand
programs using a *systematic, iterative approach*

Want to learn “how to think” about programming



Why study programming languages?

- Learning a programming “language” is superficial
 - We want to learn **how to program in a specific paradigm**
 - Study the roots of programming languages to build a mental model of big ideas that unify different paradigms and languages
 - Adapt our skills to new languages as new languages appear

Course Objective

The main goal of this course is to teach you to **write completely correct code** that you can clearly explain and easily understand

We do this through **four coding projects**

Four programming **exercises** (groups up to 3)

Two written midterms and a final (lowest drops)

Also several **homeworks** (mostly end of semester)

Attendance Quizzes

Starting the second week of the semester, we'll start having regular (once every day or two) attendance quizzes. Each quiz is worth .5%, and you can get at most 5%. There will be at least 10 quizzes (probably more like 15-18). Quizzes will be given randomly roughly once a week during lecture.

Quizzes will be conducted via online portal (Socrative), bring a laptop/phone to class. You can also turn in on paper.

Instructors

Kris Micinski (4th year asst. prof here @ SU)

Neda Abdolrahimi (PhD student @ SU)

Kris office hours:

- One hour before class on Tuesdays
- One hour after class on Thursdays

Neda office hours:

- Monday mornings, instead of labs
- Others TBD

No Labs on Mondays

We will not be holding Monday morning labs, instead we will reserve the time for ad-hoc office hours. We reserve the right to use scheduled lab sessions for exceptional circumstances. If you want extra help on Mondays, please get in touch with the TA.

Syllabus

Most up-to-date syllabus always available at:

<https://kmicinski.com/cis352-s25/syllabus>

Grading

- 50% Autograder
 - 4 projects (10% each), 4 exercises (group, 2.5% each)
- 40% Exams
 - 2 midterms, a final—each worth 20%, lowest of all drops
 - Final can be skipped if you are happy with your grade
- 5% Attendance quizzes
 - You need to show up at 10/~15 participation quizzes
- 5% Homework
 - We'll have a small number of homework sets

Projects

This course has projects (with **deadlines**) that are assigned and graded via an **autograder**

<https://autograder.org>

You are expected to use the **Git interface** to the autograder;
Autograder credentials will be sent out by the **first week**

Prepping for the Course (Homework)

Neda will show an example usage of the Autograder on Thursday (second day of class). For now, you should:

- Identify a terminal application you are comfortable using, you will need to use the command line for this class.
 - If you don't know it, this is an explicit skill we require you learn
 - At least the small amount of knowledge you need to run the grading scripts and submit projects via Git
- Download Dr. Racket
 - Go to "Help -> Configure Racket for Command Line"
 - Install git on your machine
 - Make sure Python3 is installed on your machine

Academic Integrity

No collaboration on code is allowed for projects—don't send / show / ... anyone your code. Don't **post** any project code > 3 lines

The autograder employs elaborate measures that compare code (syntactically and semantically) to identify potential collaboration, then TAs and I check manually

“Hard coding” answers (for projects, i.e., recognizing specific inputs and providing correct outputs) is also an AI violation

ChatGPT Policy

ChatGPT and LLM-based technology (Copilot, etc...) have serious potential to accelerate your programming skill, but it must be used carefully in this course

In short, you can use ChatGPT to study material if you'd like, and you can use it for the **exercises only, but not use code it generates on projects**

Start **early** on projects

We **try** to make projects sync up with the material presented at the corresponding time in the course

Biggest indicator of success in the course is whether students are on-track with projects—try to never get behind; it becomes hard to catch up.

Project Grading

- ▶ Each project is graded on a percent scale; your grade is the % of tests that pass (18/20 tests passing = 90%)
- ▶ Projects always due at 11:59PM Syracuse time
- ▶ Projects up to 72 hours after deadline—15% penalty (max 85%)
- ▶ Projects up to end of course—25% penalty
- ▶ I.e., you can, *in principle*, always get a 75%

Exams

- There will be a **two midterms** and a **final**
- Both midterms **in-class** and **written**
- Allowed one letter-sized (**single-sided**) note sheet
- I will release a practice midterm with the same question titles several days before both midterms; we will work it in class
- Final exam is slightly longer, but replaces the lower of your two midterm grades
- Each exam 20%, lowest drops (40% total from exams)

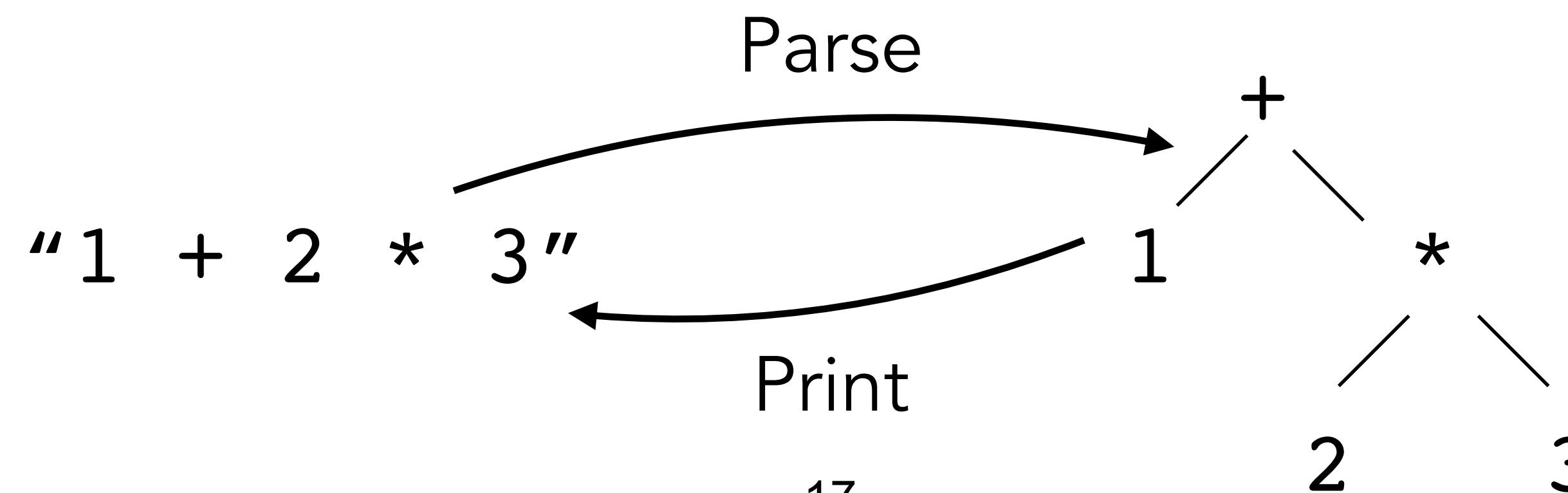
Syntax

A language's physical form, its identifiers and grammatical structure, is called its **syntax**

When we talk about programs, we often represent them as an **abstract** representation (e.g., an "abstract-syntax tree")

Tokenization and parsing is the task of turning raw syntax (stream of tokens) into an abstract representation

We will not cover parsing much



Semantics

PLs are unlike natural language—we need them to have a *precise, unambiguous* meaning

PLs have some systematically-defined meaning (semantics)

This can take several forms:

- ➊ Reference interpreter / compiler
- ➋ Written specification
- ➌ Machine-checked formal proof

Semantics

In this class we will mainly learn about semantics by building **interpreters**, though we will also speak of other kinds of semantics (e.g., the static semantics of type theory)

S

Racket Basics

CIS352

Kris Micinski



Racket

- **Dynamically-Typed:** variables are untyped, values typed
- **Functional:** Racket emphasizes functional style
 - Compositional—emphasizes black-box components
 - Immutability—requires automatic memory management
- **Imperative:** allows data to be modified, in carefully-considered cases, but doesn't emphasize "impure" code

Racket

- **Object-oriented:** Racket has a powerful object system
- **Language-oriented:** Racket is really a language toolkit
- **Homoiconic:** the same structure used to represent **data** (lists) is also used to represent **code**

Example

$\left(\begin{array}{c} (\lambda(x) (x x)) \\ (\lambda(x) (x x)) \end{array} \right)$

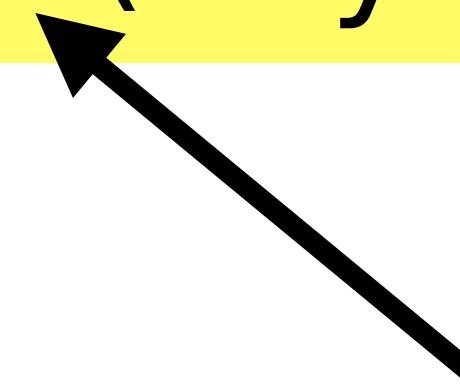
Calculating the slope of a line in Racket

```
(define (calculate-slope x0 y0 x1 y1)
  (/ (- y1 y0) (- x1 x0)))
```

Example

$$\begin{pmatrix} (\lambda(x) (x\ x)) \\ (\lambda(x) (x\ x)) \end{pmatrix}$$

```
(define (calculate-slope x0 y0 x1 y1)
  (/ (- y1 y0) (- x1 x0)))
```



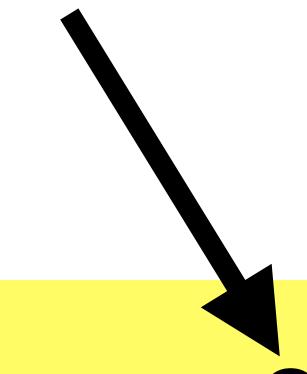
Prefix notation

Example

$$\begin{pmatrix} (\lambda(x) (x\ x)) \\ (\lambda(x) (x\ x)) \end{pmatrix}$$

Functions defined via prefix notation, too

```
(define (calculate-slope x0 y0 x1 y1)
  (/ (- y1 y0) (- x1 x0)))
```



Example

$$\begin{pmatrix} (\lambda(x) (x\ x)) \\ (\lambda(x) (x\ x)) \end{pmatrix}$$

Calls to user-defined functions also in prefix notation

```
(define (calculate-slope x0 y0 x1 y1)
  (/ (- y1 y0) (- x1 x0)))
// C - calculate-slope(0,0,3,2);
(calculate-slope 0 0 3 2)
```

Example

$\left(\begin{array}{c} (\lambda(x) (x\ x)) \\ (\lambda(x) (x\ x)) \end{array} \right)$

Note: preferred style puts closing parens at end of blocks

```
(define (calculate-slope x0 y0 x1 y1)
  (/ (- y1 y0) (- x1 x0))))
```

```
(calculate-slope 0 0 3 2)
```



Basic Types

- **Numeric tower.** Numeric types gracefully degrade
 - E.g., $(\ast (/ 8 3) 2+1i)$ is $16/3+8/3i$
 - Note that $2+1i$ is a **literal** value, as is 2.3
- **Strings and characters** (“foo” and $\#\backslash a$)
- **Booleans** ($\#t$ and $\#f$) including logical operator (e.g., or)
 - Note that operators “short circuit”

Basic Types contd.

- **Symbols** are interned strings ‘foo
 - Implicitly only one copy of each, unlike (say) strings
 - Impact on space / memory usage
 - The #<void> value (produced by (void))

Exercise



Compute the sum of the following:

- $\frac{2}{3}$ and 1.5
- $3+8i$ and $3i$
- 0 and positive infinity ($+\inf. 0$)

Exercise



Compute the sum of the following:

- (+ 2/3 1.5)

2.1666666666666665 (N.B., result is **inexact**)

- (+ 3+8i 0+3i)

3+11i

- (+ 0 +inf.0)

+inf.0