Intro to Racket
Don’t worry — this is still CS 42!

Compute the dot product:

\[(1 \ 2 \ 3) \cdot (6 \ 5 \ 4)\]

Definition of dot product:

\[a \cdot b = (a_1 \times b_1) + (a_2 \times b_2) + \ldots + (a_n \times b_n)\]
Real assembly language

Google Chrome:
(__TEXT,__text) section
0000000100000ef0        pushq   $0x0
0000000100000ef2        movq    %rsp, %rbp
0000000100000ef5        andq    $-0x10, %rsp
0000000100000ef9        movq    0x8(%rbp), %rdi
0000000100000efd        leaq    0x10(%rbp), %rsi
0000000100000f01        movl    %edi, %edx
0000000100000f03        addl    $0x1, %edx
0000000100000f06        shll   $0x3, %edx
0000000100000f09        addq    %rsi, %rdx
0000000100000f0c        movq    %rdx, %rcx
0000000100000f0f        jmp     0x100000f15
0000000100000f11        addq    $0x8, %rcx
0000000100000f15        cmpq    $0x0, (%rcx)
0000000100000f19        jne     0x100000f11
0000000100000f1b        addq    $0x8, %rcx
0000000100000f1f        callq   _main
0000000100000f24        movl    %eax, %edi
0000000100000f26        callq   0x100000f46          ## symbol stub for: _exit
0000000100000f2b        hlt
0000000100000f2c        nop
0000000100000f2d        nop
0000000100000f2e        nop
0000000100000f2f        nop
How’s it going?

(1) The pace of this class is...
   1 = way too slow; 4 = just right; 7 = way too fast

(2) I’m learning a lot in CS 42.
   1 = strongly disagree; 4 = neither agree nor disagree; 7 = strongly agree

(3) CS 42 is interesting.
   1 = strongly disagree; 4 = neither agree nor disagree; 7 = strongly agree

(4) I know I can get help / support from (e.g., Prof. Ben, the grutors, Piazza), if I need it.
   1 = strongly disagree; 4 = neither agree nor disagree; 7 = strongly agree

(5) When it comes to workload, so far, this is my hardest course this semester.
   1 = strongly disagree; 4 = neither agree nor disagree; 7 = strongly agree

(6) I find the “sketches” helpful.
   1 = strongly disagree; 4 = neither agree nor disagree; 7 = strongly agree
CS 42: Principles (& Practice) of Computer Science
The Principles in CS 42

1. **Theory of computation • Machines** (~4 weeks)
   What do we mean by *computer*?
   Programs $\iff$ Data

2. **Functional programming** (~4 weeks)
   Recursion
   Functions $\iff$ Data

3. **Problem-solving techniques** (~4 weeks)
   How do we match a problem with its solution?
   How good is a solution?
   Algorithms + Data Structures

4. **Object-oriented programming** (~3 weeks)
   How do we structure a program so that it can grow and change?

5. **Logical programming** (~1 week)
   How does language influence thought?
What is “Functional Programming”?

Read and discuss: Backus paper

What’s something you agree with?

What’s something you disagree with?

What’s something you don’t (yet) understand?

Why do you think the highlighted text is highlighted?

Can you sum up the reading in at most six words?
Functional programming
A working definition

No mutable state (no “assignments”).
It’s all about evaluating expressions.
Functions help us manage complexity.

Say more “what”; say less “how”.
Prior experience: programming languages

- Assembly
- Python
- Racket
- Java
- Prolog

none  ->  lots

soon  ->  later
How do we learn a new programming language?

What kinds of things do we want to know first?

What does it feel like as you learn a new PL?
Why Racket?
Racket—a functional programming language

A working definition

Variables

Functions

Say more “what”; say less “how”.
Dr. Racket
an Integrated Development Environment (IDE) for Racket

Run the program!

```
1  #lang racket

boilerplate: the version of Racket we’re using

“definitions” (i.e., programs) go here

Welcome to DrRacket, version 6.2.1 [3m].
Language: racket [custom]; memory limit: 128 MB.
```

“interactions” go here
Racket: “primitive” values

- integers
- booleans
  only \texttt{false} is \texttt{false}
- real numbers
- strings
- (lots more, non-primitive values)
Welcome to DrRacket, version 6.6 [3m].
Language: racket, with debugging; memory limit: 128 MB.

> 1
1
> 2
2
> true
#t
> false
#f
> 3.14
3.14
> "platypus"
"platypus"
>
Racket: operations (s-expressions)

\((op \ arg_1 \ arg_2 \ ... \ arg_n)\)

- Rules:
  - the operation always comes first
  - its arguments (if there are any) follow the operation
  - no commas between arguments
  - everything goes between parentheses

- Common mistakes:
  - forgetting parentheses
  - rational vs. integer division (/ vs. quotient)
  - equality (= vs. equal?)
Welcome to DrRacket, version 6.2.1 [3m].
Language: racket [custom]; memory limit: 128 MB.

> (+ 1 1)
2
> (+ 1 (* 2 3))
7
> (* 1 (+ 2 3))
5
> (/ 1 2)
1/2
> (quotient 1 2)
0
> (/ 1.2)
0.5
> (= 1 1)
#t
> (= 1 2)
#f
> (= "a" "b")
してください: number?
given: "a"
argument position: 1st
other arguments...
> (equal? "a" "b")
#f
> (equal? "a" "a")
#t
>