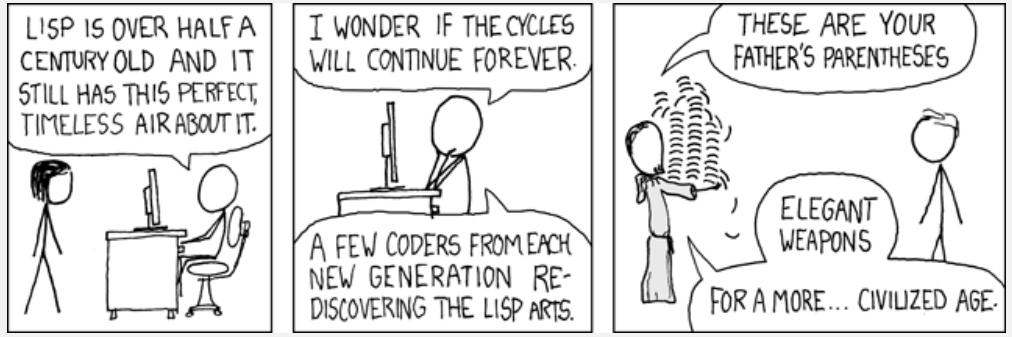
CS450 (section 2)

High Level Languages

UMass Boston Computer Science

Monday, September 9, 2024



Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I

John McCarthy, Massachusetts Institute of Technology, Cambridge.

April 1960

1 Introduction

A programming system called LISP (for LISt Processor) has been developed for the IBM 704 computer by the Artificial Intelligence group at M.I.T. The system was designed to facilitate experiments with a proposed system called the Advice Taker, whereby a machine could be instructed to handle declarative as well as imperative sentences and could exhibit "common sense" in carrying out its instructions.

LISP coming back?

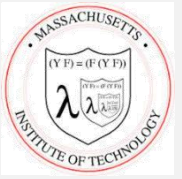
- Programs are **expressions** (not sequences of instructions!)
- S-expression syntax
  - "code is data, data is code"
  - `(list + 1 2)` is both program and list of "symbols"
- Invented: `if-then-else`, `lambda`, recursion, gc (no ptrs), `eval`

Lisp

(First "high-level" language)

Clojure

Scheme



Game Oriented Assembly Lisp (GOAL)	2000s	Andy Gavin	Video game programming language developed by Andy Gavin and the *Jak and Daxter* team at Naughty Dog; written using Allegro Common Lisp; used in developing the full game series

Racket

PAUL GRAHAM

BEATING THE AVERAGES

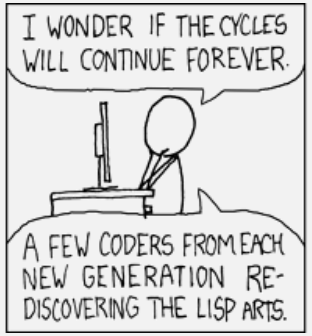
Want to start a startup? Get funded by Y Combinator. Y

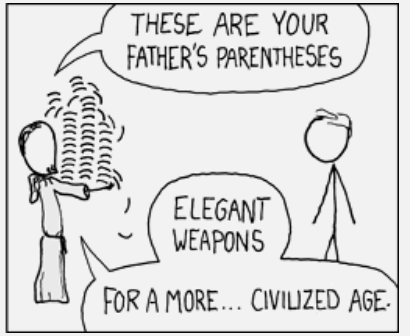
(This article is derived from a talk given at the 2001 Franz Developer Symposium.)

In the summer of 1995, my friend Robert Morris and I started a startup called Viaweb. Our plan was to write software that would let end users build online stores. What was novel about this software, at the time, was that it ran on our server, using ordinary Web pages as the interface. yahoo! shopping

Another unusual thing about this software was that it was written primarily in a programming language called Lisp. It was one of the first big end-user applications to be written in Lisp, which up till then had been used mostly in universities and research labs.

Lisp is worth learning for the profound enlightenment experience you will have when you finally get it; that experience will make you a better programmer for the rest of your days, even if you never actually use

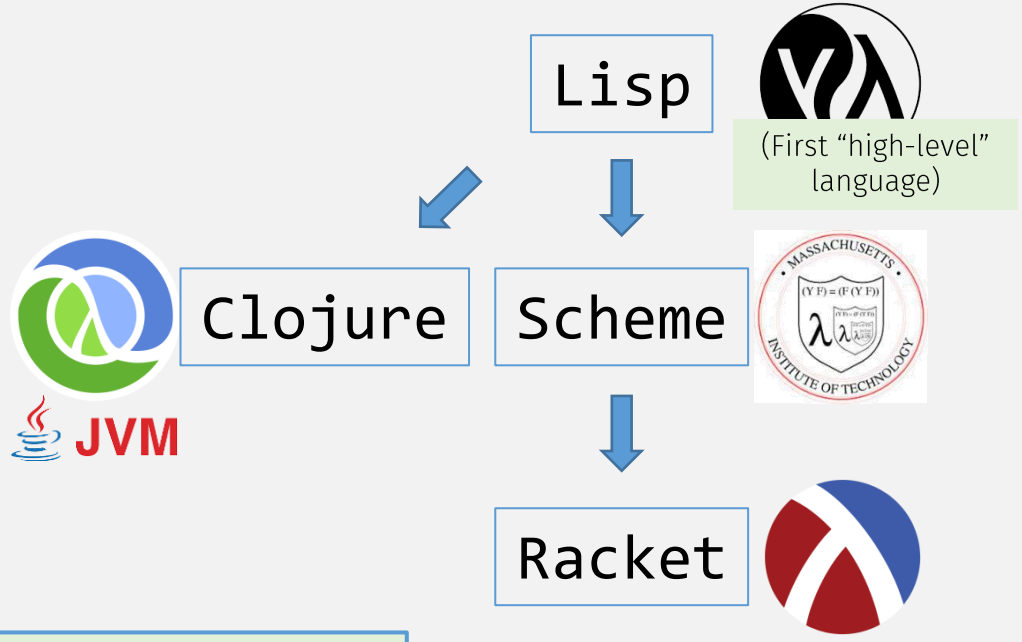




Logistics

- HW 0 in
  - ~~due: Mon 9/9 12pm (noon) EST~~
- HW 1 out
  - due: Mon 9/16 12pm (noon) EST
- Course web site:

  `https://www.cs.umb.edu/~stchang/cs450/f24`
- Add / drop ends tomorrow (Tue 9/10)



Today's Focus: Being Ready For the Course

I don't know what _____ is …

… and at this point, I'm too afraid to ask!

(don't be this person)

Racket

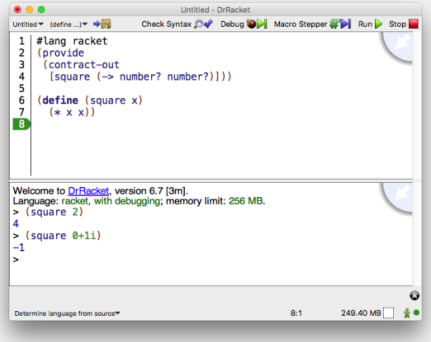
Web: `racket-lang.org`

Download: `download.racket-lang.org`

- Linux: `https://launchpad.net/~plt/+archive/ubuntu/racket`

Version: **8.6+**

IDE: **DrRacket** (easiest)

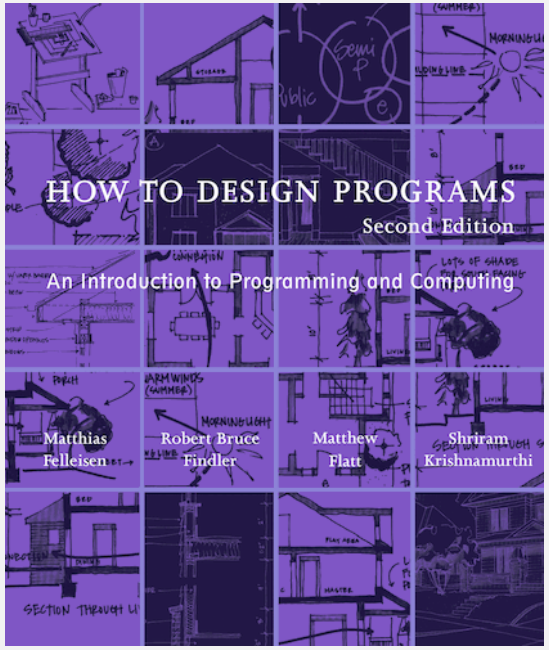


Docs: `docs.racket-lang.org`

forum:

- Hw help: **piazza**
- General: `racket.discourse.group`

Textbook: How to Design Programs 2nd ed.



htdp.org

Git and Github

Did you …

- Create `Github account`?
- Install `git client`?
  - GUI or command line ok
- Learn basic `git` commands?
  - `Clone, push, pull, fork, branch`

Other Logistics

- Piazza
  - Ask all hw questions here (not over email)
  - Faster response
  - Helps other students
- Gradescope
- Read course web page?

`https://www.cs.umb.edu/~stchang/cs450/f24`

Racket (Very) Basics

- File extension: `.rkt`
- First line: `#lang racket`
  - (The HtDP Textbook uses “Student Languages” but we will not use those)
  - (Differences will be explained)
- syntax: “S-expressions”
- Comments:
  - `;` line
  - `#;` S-expression
- Identifiers:
  - everything except: `( ) [ ] { } “ , ‘ ` ;`
  - And not: `| \ #`
  - **Case sensitive!**

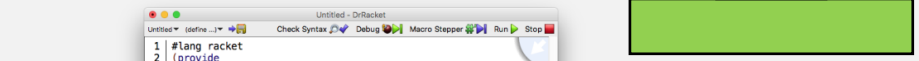
Racket Basics

- Function call: **prefix notation** (fn name first)
  - Easier to write multi-arity functions

```
(+ 1 2 3 4)
```

- (fundamental) programming model: **arithmetic**
  - But **not just numbers!**
  - **No statements!**

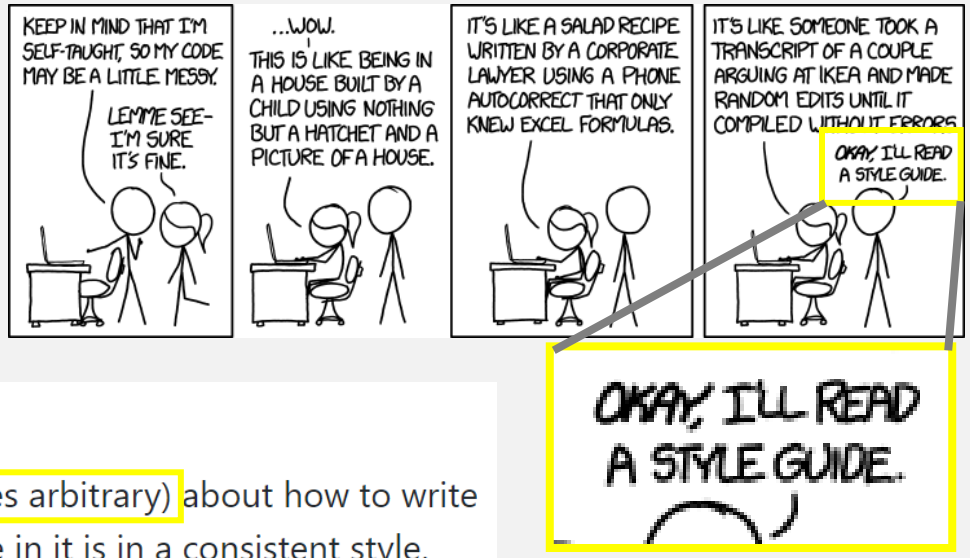
```
; delete the ith characte
(set! str
      (string-append
       (substring str 0 i
       (substring str (+
```



- Use the **REPL** (“interactions”) for basic testing!

Style

- Critical for writing readable code, e.g.



Google Style Guides

Every major open-source project has its own style guide: a set of conventions (sometimes arbitrary) about how to write code for that project. It is much easier to understand a large codebase when all the code in it is in a consistent style.

"Style" covers a [illegible] bles" to "never use exceptions." This [illegible] If you are modifying a project that or[illegible] apply to that project.

- AngularJS S[illegible]
- Common Li[illegible]
- C++ Style Guide
- C# Style Guide
- Go Style Guide
- HTML/CSS Style Guide
- JavaScript Style Guide
- Java Style Guide
- Objective-C Style Guide
- Python Style Guide

Airbnb JavaScript Style Guide() {

A mostly reasonable approach to JavaScr[illegible]

Microsoft | Learn Documentation Training Certifications Q&A Code Sampl

.NET Languages Features Workloads APIs Resources

Learn / .NET / C# guide / Fundamentals /

Common C# code conventions

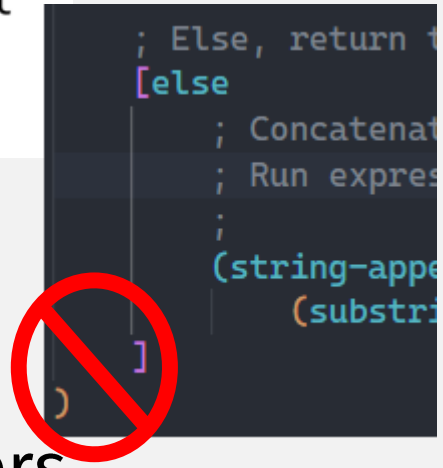
A code standard is essential for maintaining code readability, consistency, and collaboration within a development team. Following industry practices and established

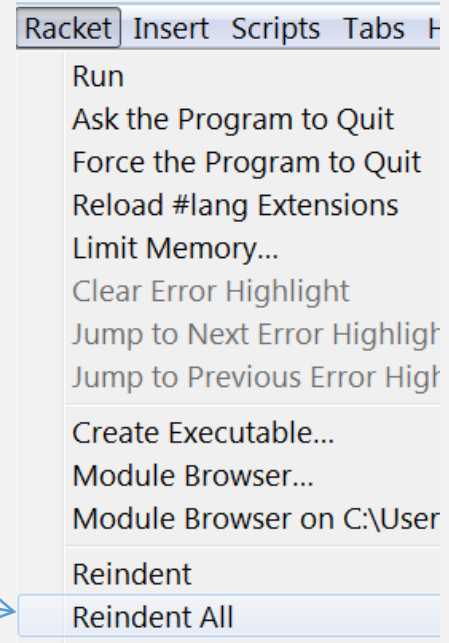
Style: This Class

https://docs.racket-lang.org/style/index.html

A few tips

- Closing parens do not get their own line
- code width 80-100 columns
- Use dashes to separate multi-word identifiers (no underscore or CamelCase):
  - `string-append`
- Use DrRacket auto-indenter





Style: Git commits

- Git commits must also be **readable** (concise and informative)

How to Write a Git Commit Message

Commit messages matter. Here's how to write them well.

The seven rules of a great Git commit message

1. Separate subject from body with a blank line
2. Limit the subject line to 50 characters
3. Capitalize the subject line
4. Do not end the subject line with a period
5. Use the imperative mood in the subject line
6. Wrap the body at 72 characters
7. Use the body to explain *what* and *why* vs. *how*

A properly formed Git commit subject line s
complete the following sentence:

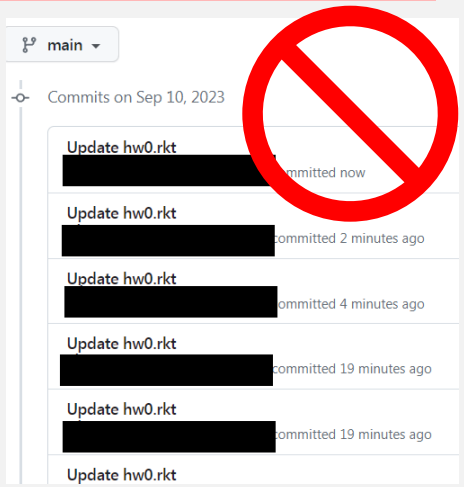
- If applied, this commit will *your subject line here*

For example:

- If applied, this commit will *refactor subsystem X for readability*
- If applied, this commit will *update getting started documentation*
- If applied, this commit will *remove deprecated methods*
- If applied, this commit will *release version 1.0.0*
- If applied, this commit will *merge pull request #123 from user/branch*

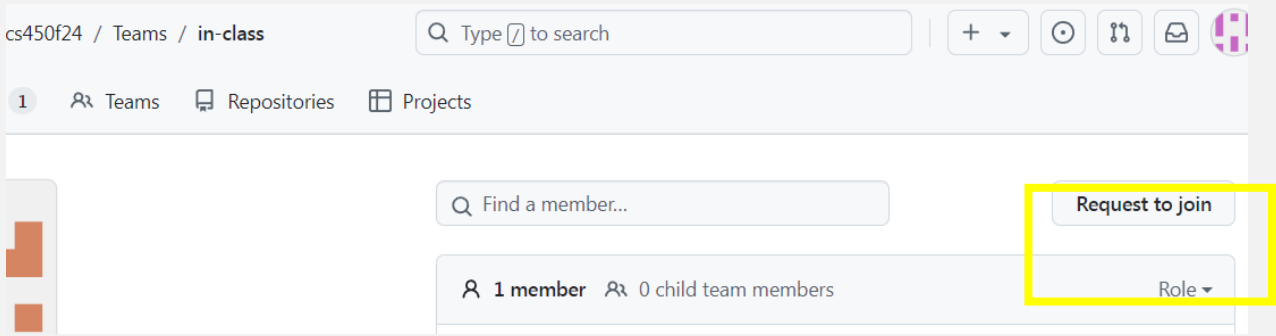
Notice how this doesn't work for the other non-imperative forms:

- If applied, this commit will ~~*fixed bug with Y*~~
- If applied, this commit will ~~*changing behavior of X*~~
- If applied, this commit will ~~*more fixes for broken stuff*~~
- If applied, this commit will ~~*sweet new API methods*~~



In-class exercise

- Using the `2htdp/image` library,
  write a Racket expression that builds "Traffic Light" image
- Put code in file named: `in-class-09-09-<Lastname>-<Firstname>.rkt`



Submitting In-class work

- Join the in-class team at:
  `https://github.com/orgs/cs450f24/teams/in-class`
- Commit your file to this repo:
  `https://github.com/cs450f24/in-class-09-09`
- (May need to **`merge`** or **`pull + rebase`**
  if someone **`push`**es before you)