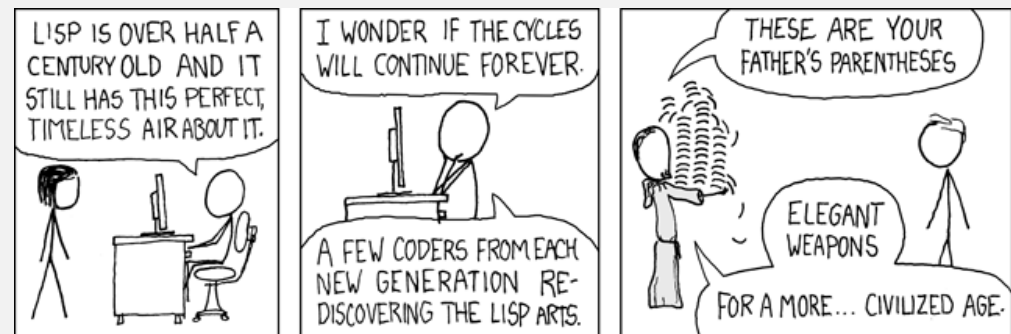


CS450

High Level Languages

UMass Boston Computer Science

Thursday, January 30, 2025



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

John McCarthy, Massachusetts Institute of Technology, Cambridge,

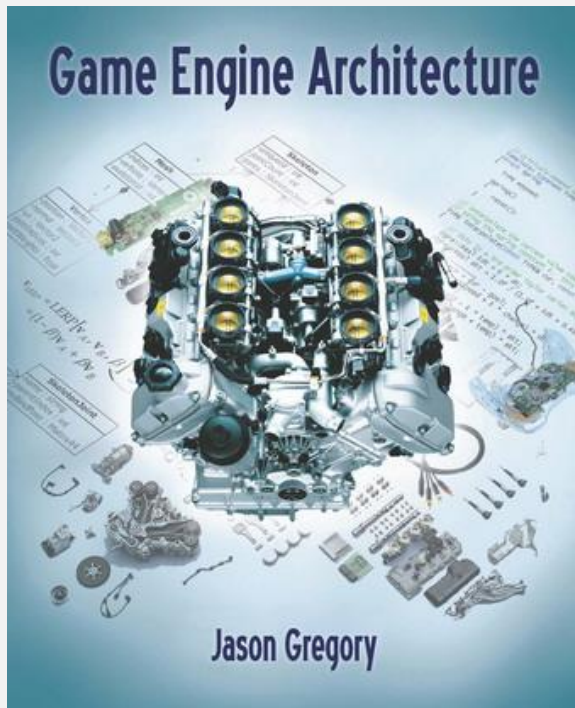
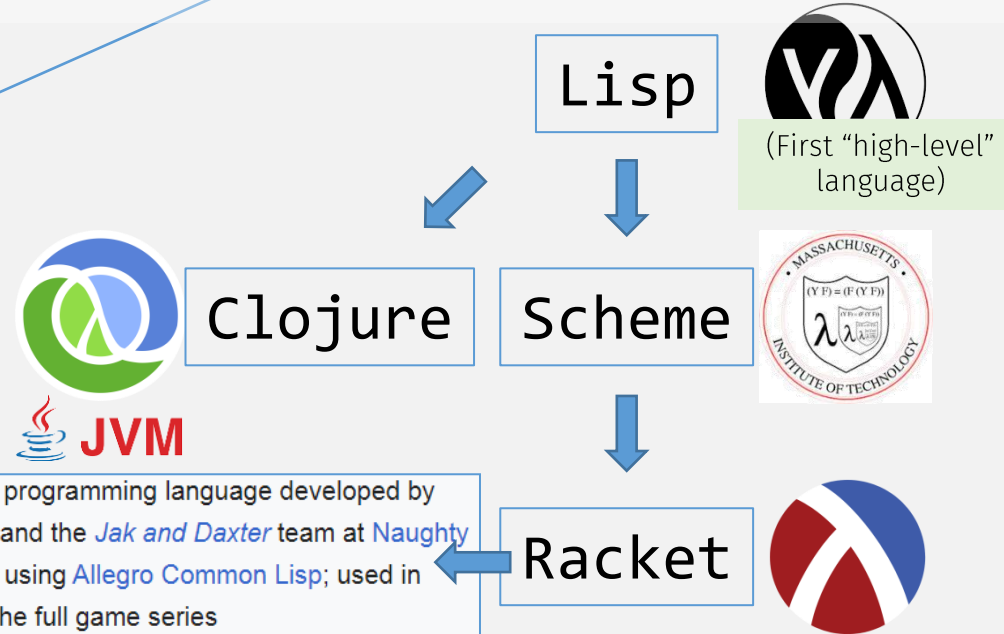
April 1960

LISP coming back?

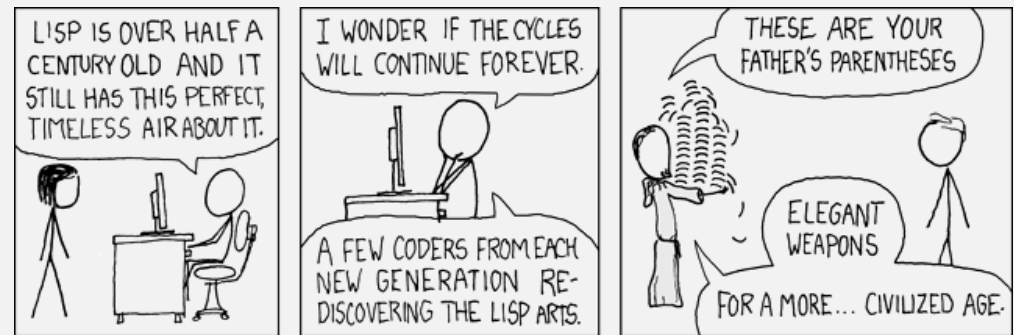
1 Introduction

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.

- Programs are **expressions** (not sequences of instructions!)
- S-expression syntax (parens!)
 - “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`



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



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



(First “high-level” language)

PAUL GRAHAM

BEATING THE AVERAGES

Want to start a startup? Get funded by [Y Combinator](#).



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

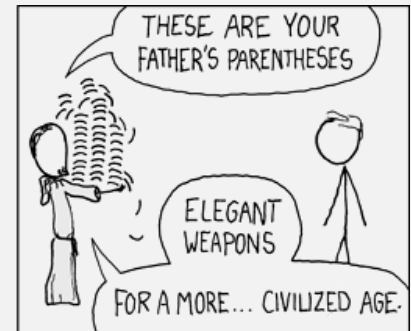
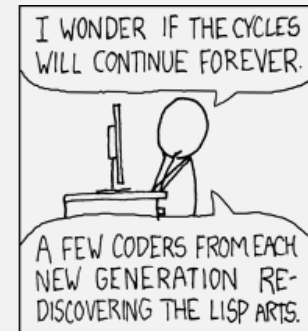
In the summer of 1995, my friend Robert Morris and I started a startup called [Viaweb](#). Our plan was to write software that would let end users build online stores. What was novel about this software, at the time, was that it ran on our server, using ordinary Web pages as the interface.




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




The Other Side ...

←  r/programmingcirclejerk • 2 yr. ago
LAUAR gofmt urself

Lisp is the worst family of languages in existence. Literally nothing can change my mind. Cool tool thingy, but you basically made a tool to teach blasphemy

It's a language that syntactically is awful. It sacrifices arbitrary elegance for practicality (human readability)

Racket does not compare to real, actual programming languages. In the real world, however, software developers use actual, practical languages like Python, C++ and to a lesser extent Javascript. The thought of using Racket never crosses their mutable variable-corrupted minds.

←  r/uwaterloo • 5 yr. ago
itsatrap12121

I HATE RACKET

UoT first year students are learning **python**

WHY DO WE HAVE TO DO **FUCKING RACKET** SHIT

RACKET IS GARBAGE

Fortunately ... this course is not about Lisp, Racket, or any other language. It is **language-agnostic!**

It's about general, high-level programming principles ... that can be used when programming in any language

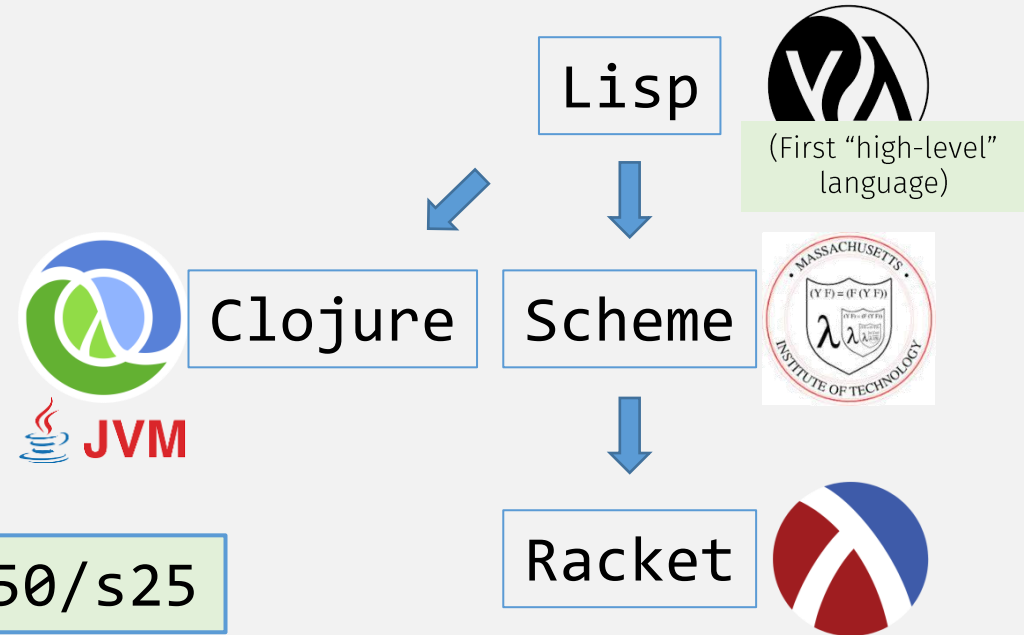
Logistics

- HW 0 out
 - due: Tue 2/4 11am EST

- Course web site:

<https://www.cs.umb.edu/~stchang/cs450/s25>

- Add / drop ends Monday 2/3



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)

```
1 #lang racket
2 (provide
3  (contract-out
4   [square (-> number? number)]))
5
6 (define (square x)
7   (* x x))
8
```

Welcome to DrRacket, version 6.7 [3m].
Language: racket, with debugging; memory limit: 256 MB.
> (square 2)
4
> (square 0+11)
-1
>

Docs: docs.racket-lang.org

forum:

- HW help: Piazza
- General: racket.discourse.group

(textbook for this course)

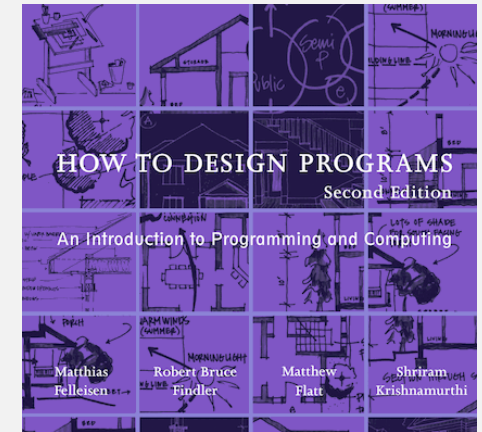
How to Design Programs, 2nd ed.

Lessons:

- How to “solve problems”, i.e., program, from scratch
- Programs are (also) for high-level communication
i.e.,
- Programs are more than “my code works”
- ... must be readable / explainable by others!

Available free at: **htdp.org**

- Can buy paper copy (make sure it's 2nd ed)



This is our rulebook!

Every org / company has its own rules for how to write clean, readable programs

Git and Github

By next Tues:

- Create Github account?
 - And tell me (in survey – must log in with umb account)
- Install git client
 - GUI or command line ok
- Learn basic git commands?
 - Clone, push, pull, fork, branch

Other Logistics

- GitHub – store HW code here
- Piazza – ask HW questions here (don't email staff)
 - Faster response
 - Helps other students
 - HW posts should be public (anonymous ok)
- Gradescope – submit HW here
- Read course web page?

<https://www.cs.umb.edu/~stchang/cs450/s25>

Grading

- **HW: 80%**
 - Weekly: in/out Tuesday (usually)
 - Approx. 12 assignments
 - Lowest grade dropped
- **Participation: 20%**
 - In-class work, lecture, office hours, Piazza
- **No exams**
- **A range: 90-100**
- **B range: 80-90**
- **C range: 70-80**
- **D range: 60-70**
- **F: < 60**

<https://www.cs.umb.edu/~stchang/cs450/s25>

Grading

- **HW: 80%**
 - Weekly: in/out Tuesday (usually)
 - Approx. 12 assignments
 - Lowest grade dropped

Evaluated on program:

- **correctness**
 - i.e., test suites
- **readability**
 - Can **someone** read and explain what it does?
 - Does **code** follow the “rules” of this class
- **understanding**
 - Can you read and/or explain what it does?

<https://www.cs.umb.edu/~stchang/cs450/s25>

Note: Autograders are for ... Graders

- (A draft version) may occasionally be released early
 - ... for your benefit ... (or detriment?) – use at your own risk
 - Not a debugging tool
- Not guaranteed to be the same as final grader
 - Official grading computed after due date
- Questions about autograder are not allowed
 - E.g., “why does my code fail the autograder?”
 - They will be ignored
- Staff has no obligation to provide an autograder
 - or answer any questions about whether code is “correct”

How Not to Ask HW Questions

- Questions code correctness is not allowed
 - E.g., “Is this code right?”
 - They will be ignored
- Vague questions are not allowed
 - E.g., “What’s wrong???”
 - E.g., “How do I start this assignment???”
- Staff has no obligation to debug code
 - E.g., “Help me find the problem in this 1000 line program!”
 - If the writer of code doesn’t understand it, then nobody else will!

How to Ask HW Questions

- Provide context
 - E.g., “I’ve tried X, Y, and Z ...”
- Provide examples
 - E.g., “Is this a valid way to write Programming Principle # 12?”
- Minimize the examples
 - No: “Are these 1000 lines correct???”
 - Yes: “I expected this line of code to evaluate to X, is my understanding ok?”
 - Yes: “What does this <single word> in the homework mean?”

Ask Clarification Questions!

- For every programming task, “correct behavior” is initially unclear
- Your first task is to make it clear
- Do not write any code until you do so
 - Otherwise, how can you possibly come up with a correct solution?
- HW questions must include an example, and should go as follows:
 - E.g., “I expect `<example expression>` to evaluate to `<answer>`.
Am I understanding the problem correctly?”
 - E.g., “I expect `(p/lst (list 1 2 3 4 5))` to evaluate to `678`.
Am I understanding the problem correctly?”

Grading

- **Participation: 20%**

- In-class work, lecture, office hours, Piazza

- This is an **in-person class**
 - ... with frequent in-class work

- **Lectures are recorded**
 - ... for emergencies or review
 - availability not guaranteed

- **Active participation**
 - ... correlated with grade

- Participation can only help

<https://www.cs.umb.edu/~stchang/cs450/s25>

Late HW

- Is bad ...
 - Grades get delayed
 - Can't discuss solutions
 - You fall behind!
- Late Policy: **3 late days** to use during the semester

HW Collaboration Policy

Allowed

- Discussing HW with classmates (but must cite)
- Using other resources, e.g., youtube, other books, etc.
- Writing up answers on your own, from scratch, in your own words / code

Not Allowed

- Submitting someone else's answer
- It's still someone else's answer if:
 - variables are changed,
 - words are omitted,
 - or sentences rearranged ...
- Using sites like Chegg, CourseHero, Bartleby, Study, etc.
- Using AI bots like ChatGPT, Copilot, Claude, DeepSeek, etc.

Honesty Policy

- 1st offense: zero on problem
- 2nd offense: zero on hw, reported to school
- 3rd offense+: F for course

Regret policy

- If you self-report an honesty violation, you'll only receive a zero on the problem and we move on.

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

This position must be an
(arithmetic expression that
evaluates to a) function value

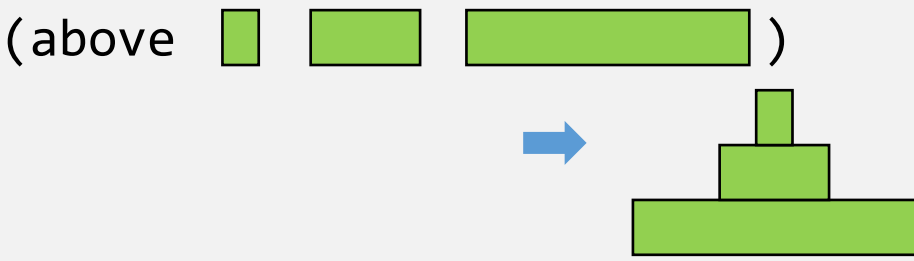
- Function call: **prefix notation** (fn name first)
 - Easier to write multi-arity functions

(+ 1 2 3 4)

- (fundamental) programming model: **arithmetic expressions**
 - But not just numbers!
 - When “run”, arithmetic expressions **evaluate** to an **answer** or **value**

arithmetic expressions

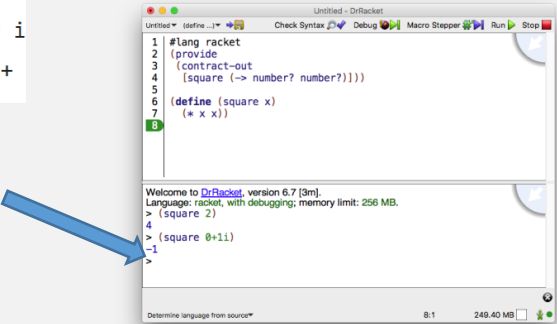
(string-append “hi” “world”)
→ “hi-world”



- No statements!
 - E.g., “assign” or “return”

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

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



Functions

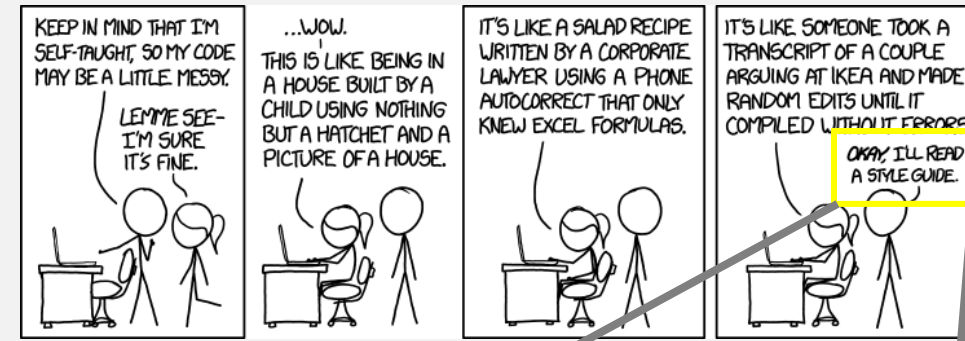
- `define` defines a function
 - The only non-expression you should use (for now)
 - `(define (fn-name arg1 arg2)
... body-expression ...)`
 - NOTE: `(define const-name expression)` defines a constant
- `lambda`
 - (anonymous) function expression
 - Function position in function call is just another expression!
 - `((lambda (x) (+ x 1)) 10)`
 - `((lambda (f) (f f)) (lambda (f) (f f)))` Warning!
- Predicates?
 - Function that **evaluates to true** or **false** when **called** or **applied**

Programs

- Programs are sequence of defines and expressions
 - One of them could be a “main” entry point
- When program is “run”, each is **evaluated** to get an **answer / value**
 - similar to “reduction” in math

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 lot of things, from "never use exceptions." This is a project that or

Airbnb JavaScript Style Guide() {

If you are modifying apply to that project.

OKAY, I'LL READ A STYLE GUIDE.

- [AngularJS Style Guide](#)
- [Common Lisp Style Guide](#)
- [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](#)

A mostly reasonable approach to JavaScript

Microsoft | [Learn](#) | [Documentation](#) | [Training](#) | [Certifications](#) | [Q&A](#) | [Code Samples](#)

[.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>

 Racket
How to Program Racket: a Style Guide

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

```
; Else, return t  
[else  
; Concatenat  
; Run expres  
; ;  
(string-appen  
  (substri  
  )  
]
```



Racket Insert Scripts Tabs F

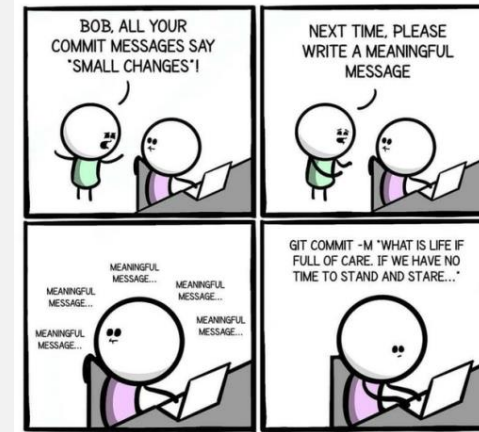
- Run
- Ask the Program to Quit
- Force the Program to Quit
- Reload #lang Extensions
- Limit Memory...
- Clear Error Highlight
- Jump to Next Error Highlight
- Jump to Previous Error High
- Create Executable...
- Module Browser...
- Module Browser on C:\User
- Reindent
- Reindent All



Style: Git commits

	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAAAA	3 HOURS AGO
○	ADKFJSLKDFJSOKLJFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.



- 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 **complete the following sentence:**

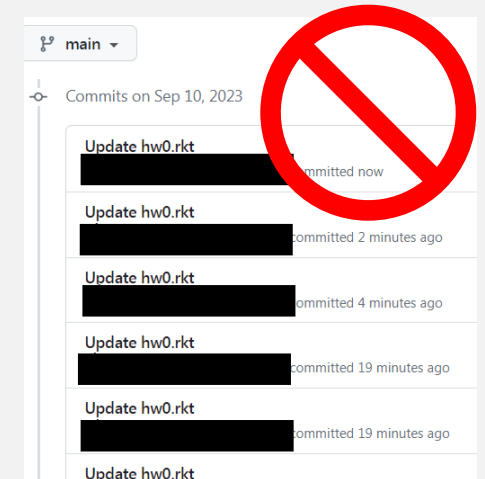
- If applied, this commit will your subject line here

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 `2htdp/image` library:
write a Racket expression that builds a “Traffic Light” image
- Put in file: `in-class-01-30-<Lastname>-<Firstname>.rkt`

Submit In-class work

- Join “in-class” team at:
<https://github.com/orgs/cs450s25/teams/in-class>
- Commit file to this repo:
<https://github.com/cs450s25/in-class-01-30>
- (May need to `merge` or `pull + rebase` if someone pushes before you)

