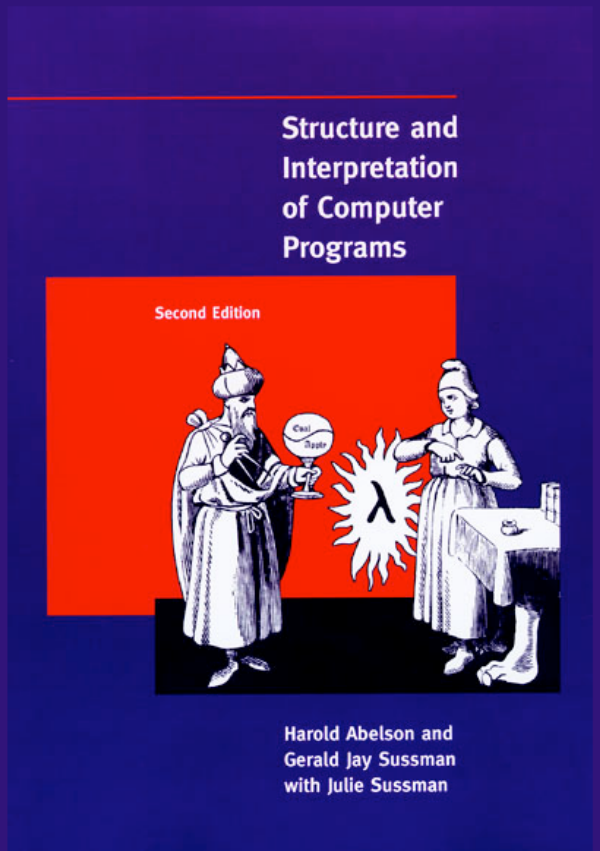
WIZARD BOOK STUDY GROUP

Welcome! - please: sound off, video on



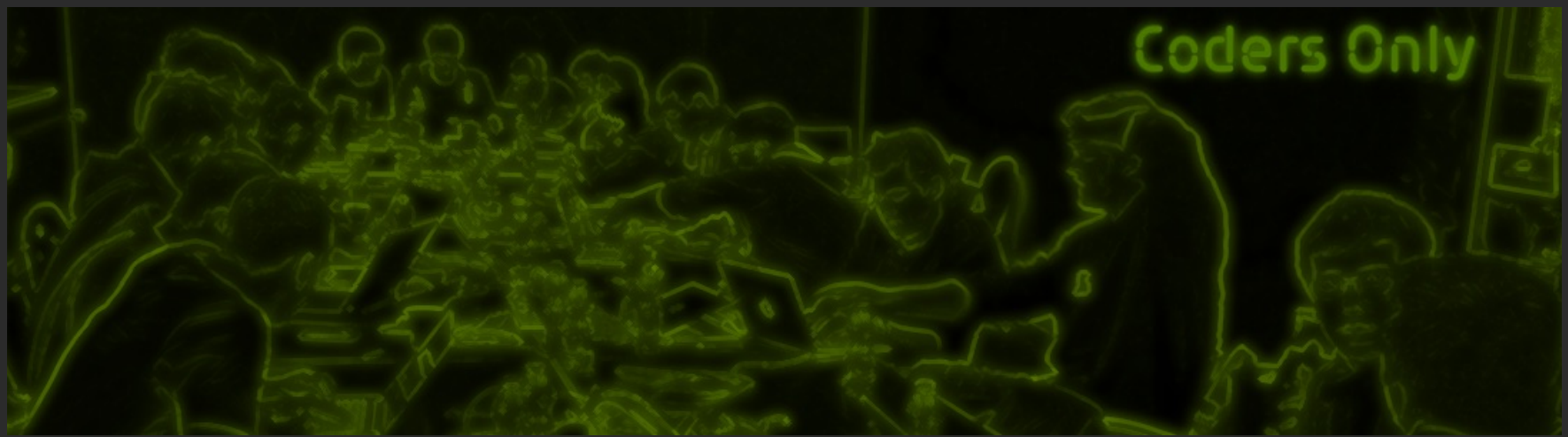
Coders Only

Studying the Wizard Book

one mind blowing idea per week

AGENDA KICK-OFF

- Who are we (country)
- Our Goal
- the book SICP
- the Berkeley lectures
- Our Repo
- Pace
- Questions & Answers



Thanks to `Coders Only` for hosting our infrastructure

- repository on https://sourcehut.org/
- videoconferencing on https://whereby.com/

ETIQUETTE

- stay on mute unless you talk
- raise your hand if you want to talk
- keep your video feed on

Generally: no discrimination

GOAL

First, we want to establish the idea that a computer language is not just a way of getting a computer to perform operations but rather that it is a novel formal medium for expressing ideas about methodology.

GOAL

*Thus, **programs must be written for people to read**, and only **incidentally** for machines to execute.*

GOAL

Second, we believe that the essential material to be addressed by a subject at this level is not the syntax of particular programming-language constructs, nor clever algorithms for computing particular functions efficiently, nor even the mathematical analysis of algorithms and the foundations of computing,

GOAL

but rather ***the techniques used to control the intellectual complexity of large software systems.***

HOW DO WE MEET THIS GOAL?

- `Abstraction` is the central theme
- with very simple building blocks (Lisp) we will build complex systems from scratch

high level of abstraction

```
application programs
high-level language (Scheme)
low-level language (C)
assembly language
machine language
architecture (registers, memory, arithmetic unit, etc.)
circuit elements (gates)
transistors
solid-state physics
quantum mechanics
```

low level of abstraction

[Alan Kay:] Yes, that was the big revelation to me — when I finally understood that the half page of code on the bottom of page 13 of the Lisp 1.5 manual was Lisp in itself.

These were "Maxwell's Equations of Software!"

This is the whole world of programming in a few lines that I can put my hand over.

evalquote is defined by using two main functions, called eval and apply. apply handles a function and its arguments, while eval handles forms. Each of these functions also has another argument that is used as an association list for storing the values of bound variables and function names.

```
evalquote[fn;x] = apply[fn;x;NIL]
```

where

```
apply[fn;x;a] =
    [atom[fn] → [eq[fn;CAR] → caar[x];
                 eq[fn;CDR] → cdar[x];
                 eq[fn;CONS] → cons[car[x];cadr[x]];
                 eq[fn;ATOM] → atom[car[x]];
                 eq[fn;EQ] → eq[car[x];cadr[x]];
                 T → apply[eval[fn;a];x;a]];
     eq[car[fn];LAMBDA] → eval[caddr[fn];pairlis[cadr[fn];x;a]];
     eq[car[fn];LABEL] → apply[caddr[fn];x;cons[cons[cadr[fn];
                                                caddr[fn]];a]]]
eval[e;a] = [atom[e] → cdr[assoc[e;a]];
     atom[car[e]] →
                 [eq[car[e],QUOTE] → cadr[e];
                 eq[car[e];COND] → evcon[cdr[e];a];
                 T → apply[car[e];evlis[cdr[e];a];a]];
     T → apply[car[e];evlis[cdr[e];a];a]]
```

pairlis and assoc have been previously defined.

```
    evcon[c;a] = [eval[caar[c];a] → eval[cadar[c];a];
                  T → evcon[cdr[c];a]]
and
    evlis[m;a] = [null[m] → NIL;
                  T → cons[eval[car[m];a];evlis[cdr[m];a]]]
```

LISP ISN'T PRACTICAL AT ALL!

- You'll be able to use these concepts everywhere independently of the language
- If you're using Python or JavaScript, you're in luck
- The future is massive parallel computations

THE BOOK

Written very concisely

- read *everything* slowly
  - Foreword, introduction, footnotes
- don't get scared by the hairy mathematical examples, we won't use them

1 BUILDING ABSTRACTIONS WITH PROCEDURES

1.1 The Elements of Programming
1.2 Procedures and the Processes They Generate
1.3 Formulating Abstractions with Higher-Order Procedures

FUNCTIONAL PROGRAMMING, RECURSION

2 BUILDING ABSTRACTIONS WITH DATA

2.1 Introduction to Data Abstraction
2.2 Hierarchical Data and the Closure Property
2.3 Symbolic Data
2.4 Multiple Representations for Abstract Data
2.5 Systems with Generic Operations

REPRESENTING SEQUENCES AND TREES

3 MODULARITY, OBJECTS, AND STATE

```
3.1  Assignment and Local State
3.2  The Environment Model of Evaluation
3.3  Modeling with Mutable Data
3.4  Concurrency: Time Is of the Essence
3.5  Streams
```

STATE AND ASSIGNMENTS, CONCURRENCY, STREAMS

4 METALINGUISTIC ABSTRACTION

```
4.1  The Metacircular Evaluator
4.2  Variations on a Scheme -- Lazy Evaluation
4.3  Variations on a Scheme -- Nondeterministic Computing
4.4  Logic Programming
```

WE BUILD A LISP INTERPRETER IN LISP

5 COMPUTING WITH REGISTER MACHINES

5.1 Designing Register Machines
5.2 A Register-Machine Simulator
5.3 Storage Allocation and Garbage Collection
5.4 The Explicit-Control Evaluator
5.5 Compilation

WE BUILD A REGISTER MACHINE AND A COMPILER

HOW DO WE DO THIS?

1. We are in a group

HOW DO WE DO THIS?

1. We are in a group
2. We take small bites

HOW DO WE DO THIS?

1. We are in a group
2. We take small bites
3. We follow the Berkeley lectures

BERKELEY COURSE SIMPLIFIES SICP

- Hairy mathematical examples avoided
- simpler constructs introduced before lists: words and sentences
- some skipping (order of chapters changed)
- Chapter 5 ignored

THE LECTURES

COURSE SUMMARY

1. FUNCTIONAL PROGRAMMING

- focus:
  - repeatable input-output behavior
  - composition of functions to layer complexity
- hidden:
  - side effect mechanisms (assignment)
  - internal control structure of procedures

2. DATA ABSTRACTION

- focus:
  - semantic view of data aggregates
- hidden:
  - actual representation in memory

3. OBJECT ORIENTED PROGRAMMING

- focus:
  - time-varying local state
  - metaphor of many autonomous actors
- hidden:
  - scheduling of interactions within the one computer
  - procedural methods within an object

4. STREAMS

- focus:
  - metaphor of parallel operations on data aggregates
  - signal processing model of computation
- hidden:
  - actual sequence of events in the computation

5. PROGRAMMING LANGUAGES

- focus:
  - provide a metaphor for computation
  - embody common elements of large groups of problems
- hidden:
  - technology-specific implementation medium
  - storage allocation, etc.

6. LOGIC PROGRAMMING

- focus:
  - declarative representation of knowledge
  - inference rules
- hidden:
  - inference algorithm

OUR REPO

- All material pre-downloaded
- all code in book and lectures, translated in Racket (.rkt)
- please tell me if I've missed anything

THE LANGUAGE

Racket (a dialect of Scheme, which is a dialect of Lisp)

STANDARDS

- Dr Racket with

```
#lang racket
(require berkeley)
```

PACE

The difficulty of the homework will determine the pace

- initial 2-3 weeks: OK?
- Wednesday or Friday?

OUR LOGISTICS

- Comprehension questions, homework questions: Discord channel
  - SICP Study Group
- Sessions: invites through Meetup

HOMEWORK:

- install Dr. Racket incl. bekerley package
- log on to our Discord chat
- read the book pages 1-31
- watch lectures 1 & 2
- read the course notes for Week 1
- do homework for Week 1
  - you will need to read word.txt

QUESTIONS & ANSWERS