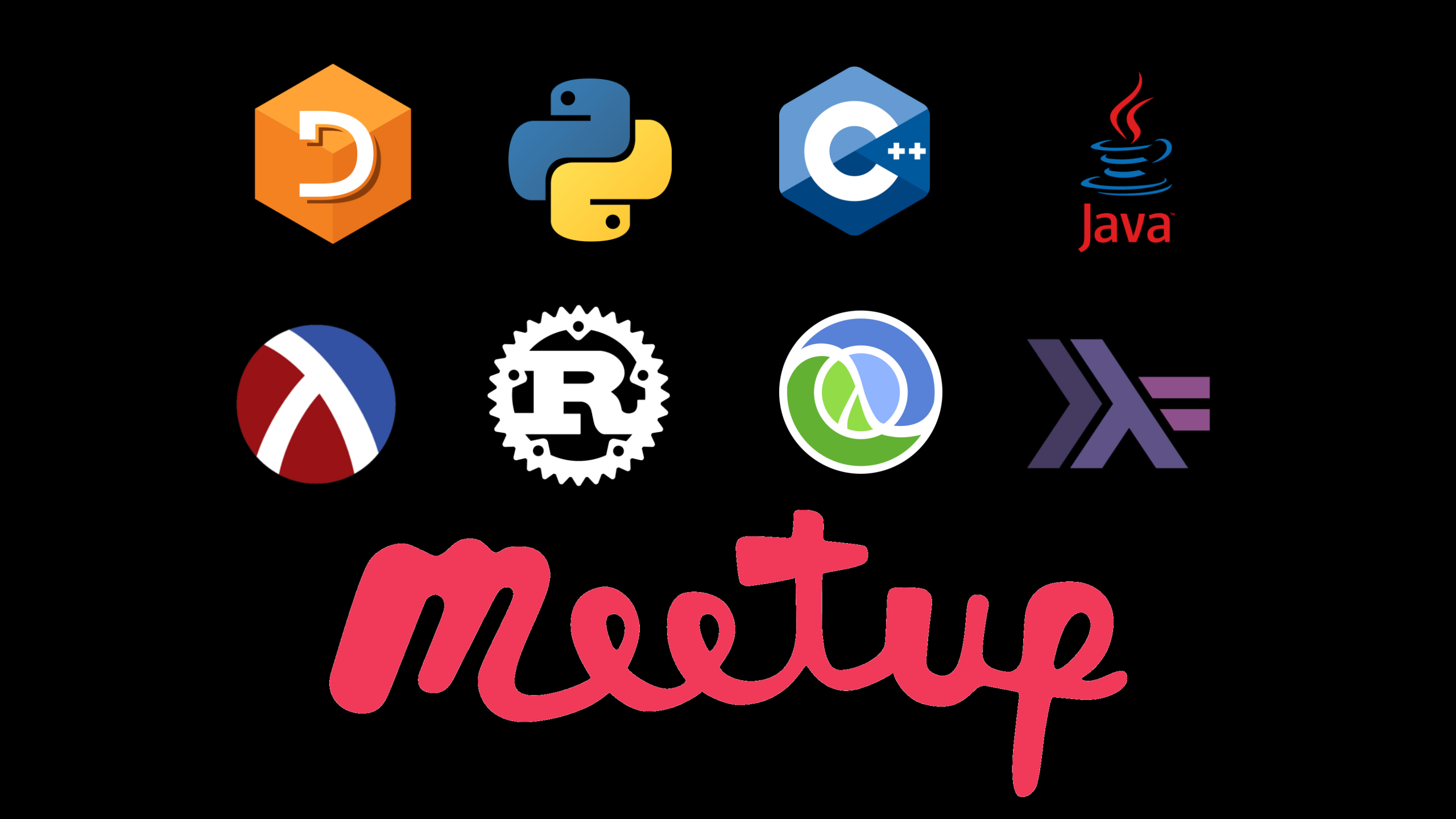
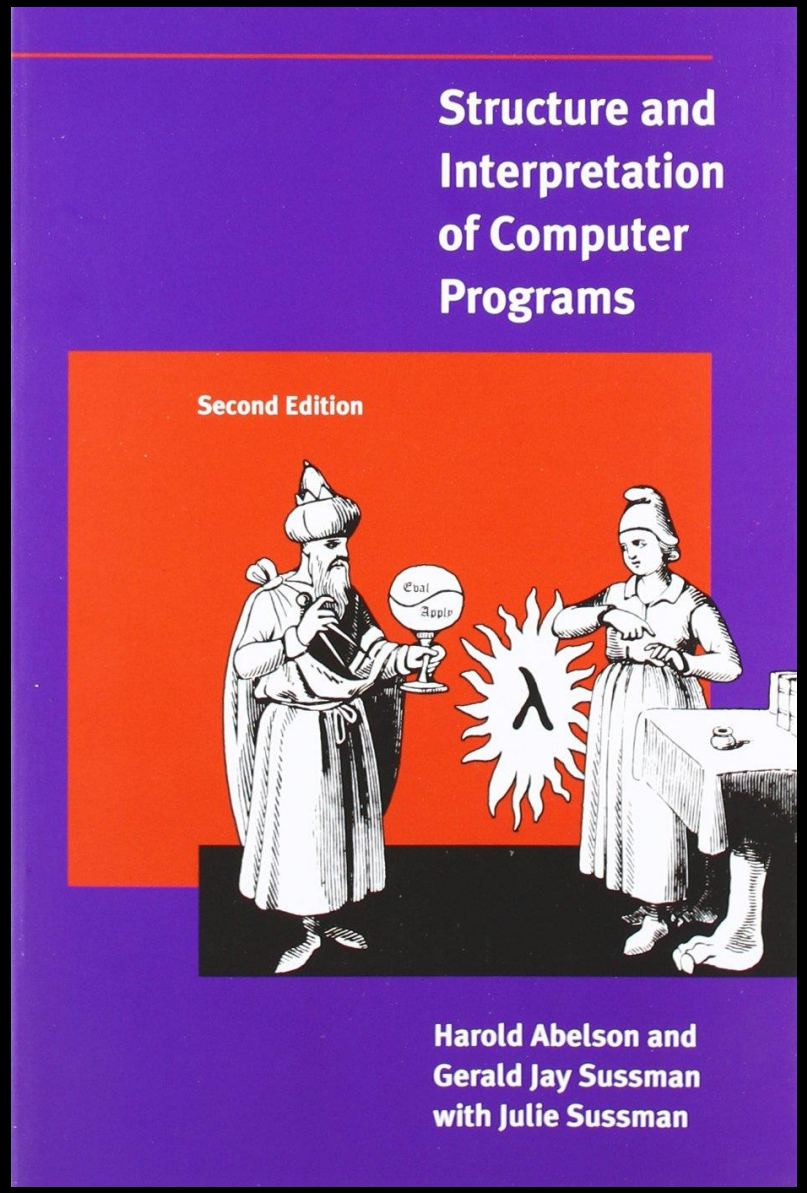

Java
meetup



Structure and Interpretation of Computer Programs

Chaper 1.1

Friendly Environment Policy

About Me (Conor Hoekstra)

- I'm a Senior Library Software Engineer for NVIDIA
  - Working on the RAPIDS AI team (http://rapids.ai)
- I am a programming language enthusiast
- Most of experience in C++
- On the ISO C++ National Body
- I love algorithms and beautiful code
- I have a YouTube channel
  youtube.com/codereport
- My online alias is `code_report`

Goal of PLVM

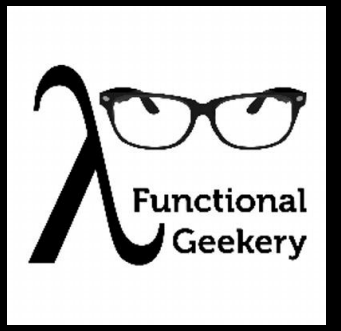
- Work through books on programming languages together
- Grow knowledge on PLs and PL:
  - principles
  - design
  - implementation
- This ultimately will lead to ability to write **code** that is more:
  - readable & expressive
  - maintainable & scalable
  - beautiful & idiomatic

Format of PLVM

- One hour meeting once a week
- ~15 min presentation at beginning
- ~45 of discussion afterwards

- I will pre-record presentation and upload to YouTube for those unable to attend

Why SICP?



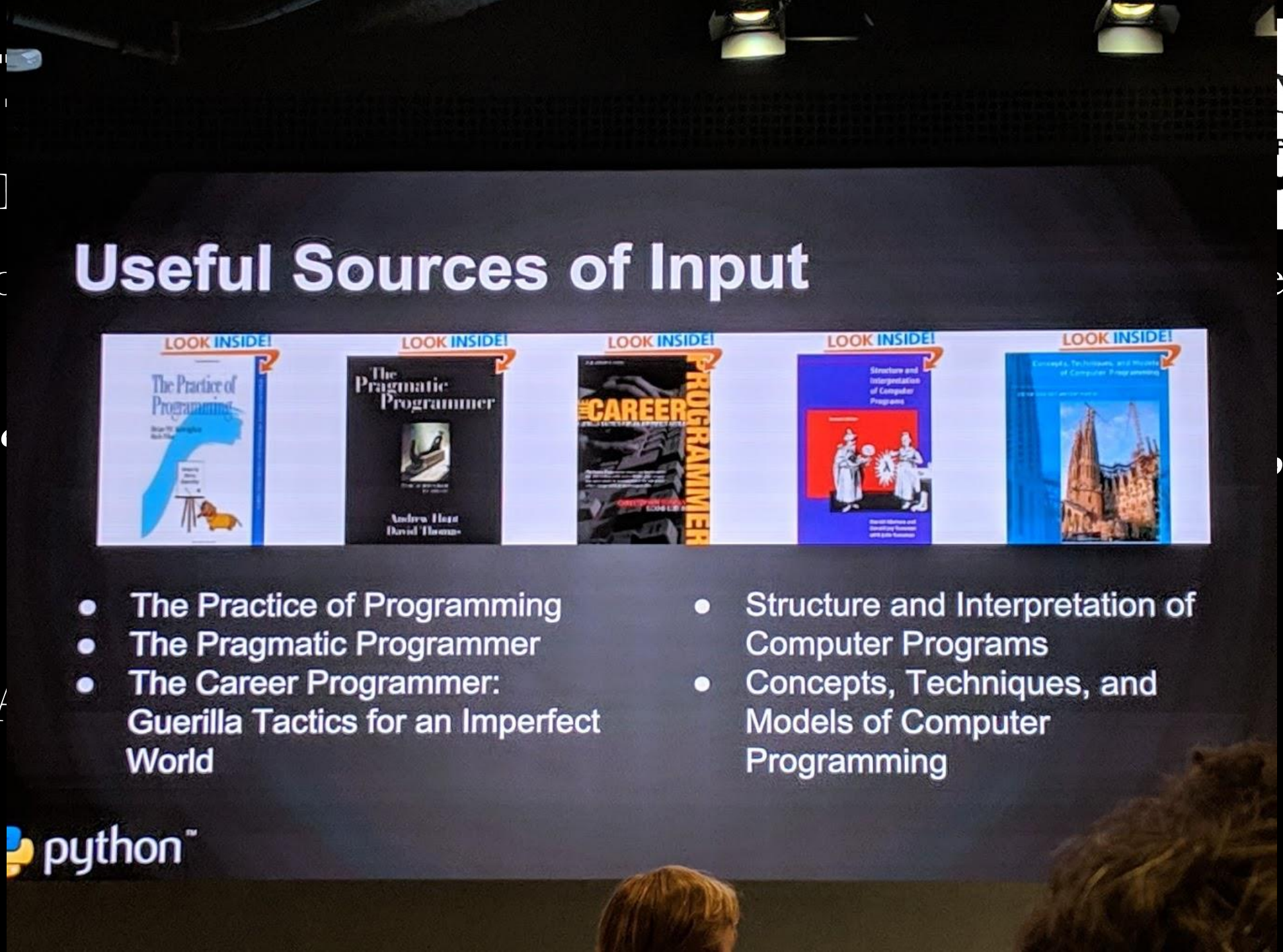
- SICP has been on my TO READ list for a long time
- Robert "Uncle Bob" Martin on Functional Geekery Episode 1

"It is pretty amazing. It was startling for me to read it, it has become a kind of bible for me"

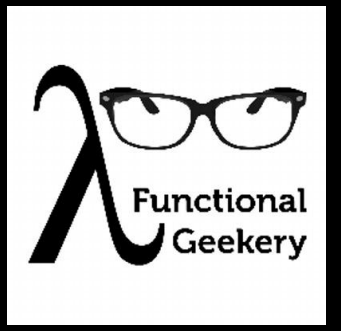
- BAyPIGgies User Group Meeting Nov 15, 2018

W

- Sl
- Rc e 1
- BA


Useful Sources of Input
LOOK INSIDE!
The Practice of Programming
LOOK INSIDE!
The Pragmatic Programmer
Andrew Hunt
David Thomas
LOOK INSIDE!
THE CAREER PROGRAMMER
LOOK INSIDE!
Structure and Interpretation of Computer Programs
LOOK INSIDE!
The Practice of Programming
The Pragmatic Programmer
The Career Programmer: Guerilla Tactics for an Imperfect World
Structure and Interpretation of Computer Programs
Concepts, Techniques, and Models of Computer Programming
python

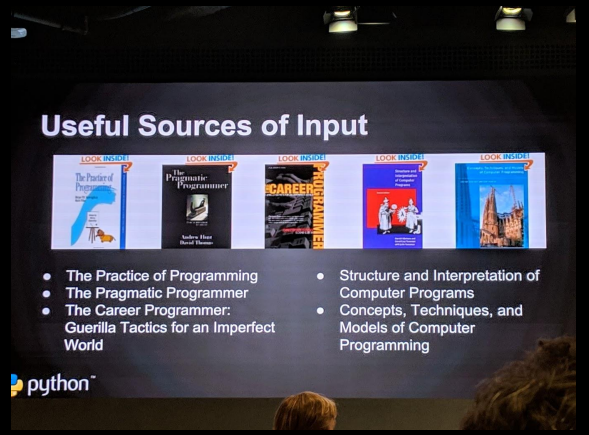
Why SICP?



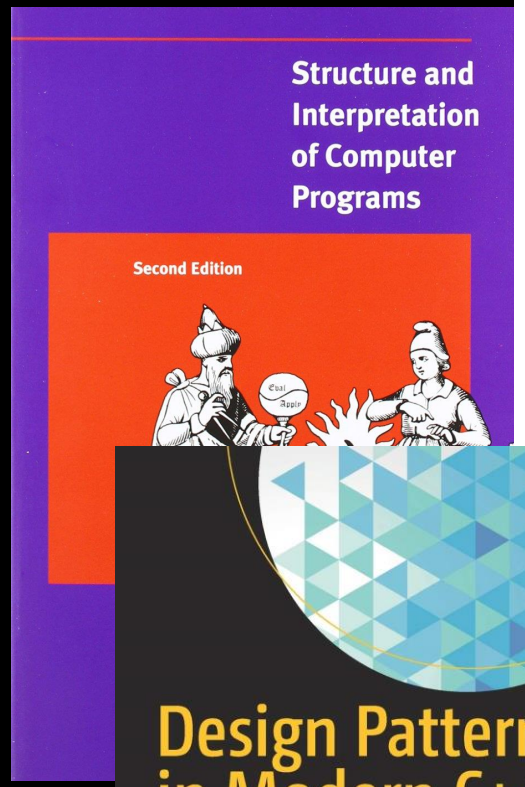
- SICP has been on my TO READ list for a long time
- Robert "Uncle Bob" Martin on Functional Geekery Episode 1

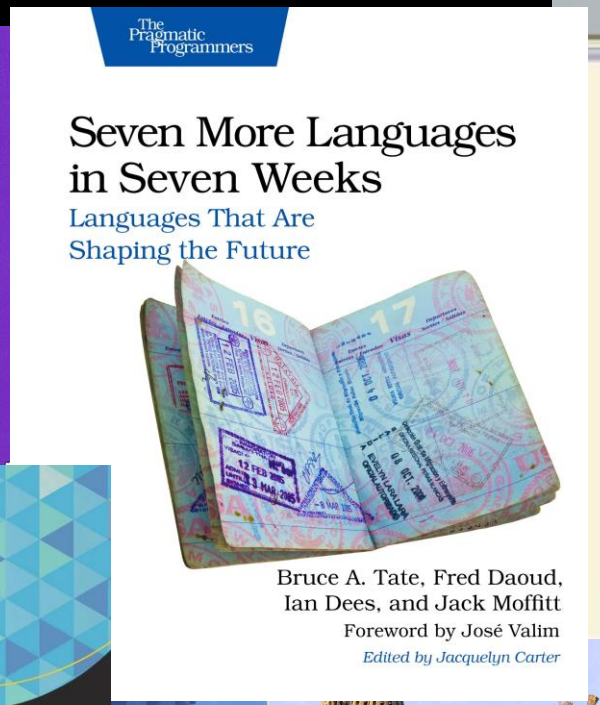
"It is pretty amazing. It was startling for me to read it, it has become a kind of bible for me"

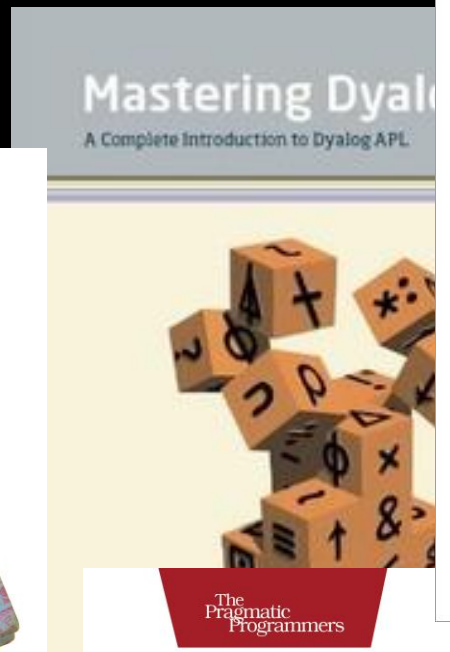
- BAyPIGgies User Group Meeting Nov 15, 2018

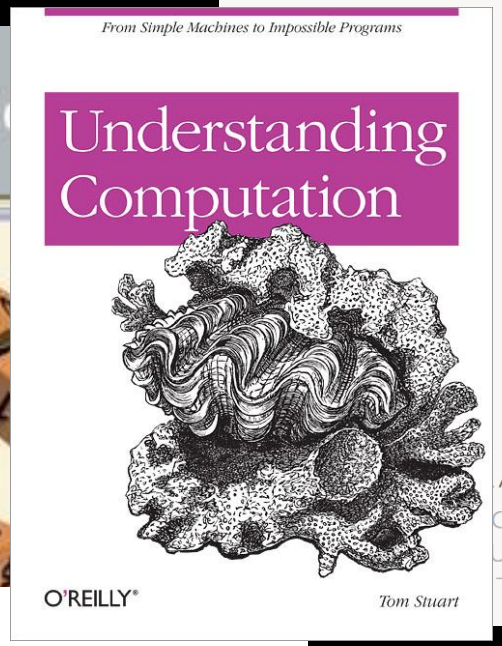


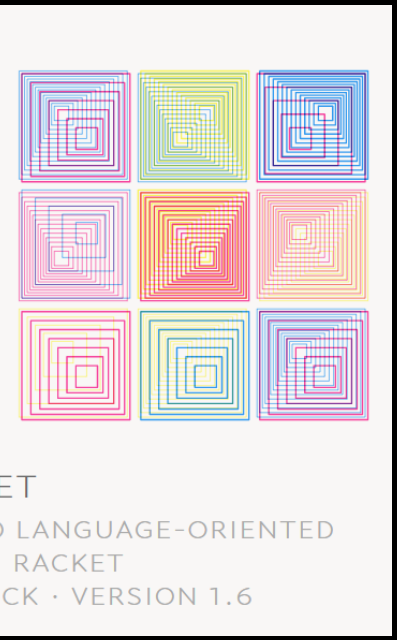
Books to read:

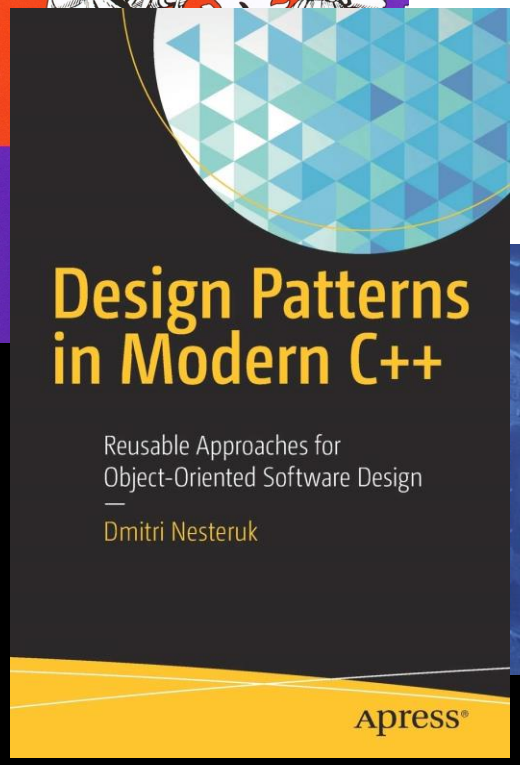


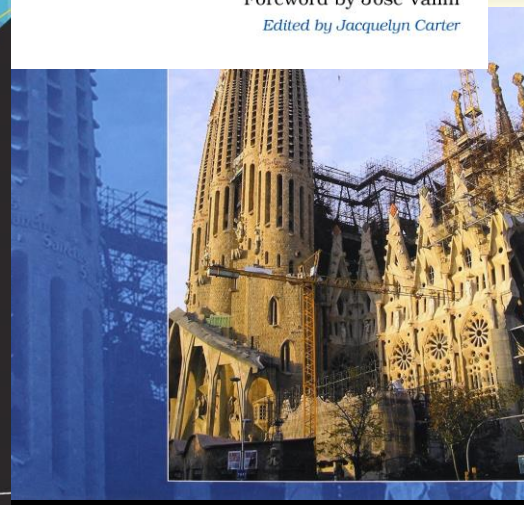


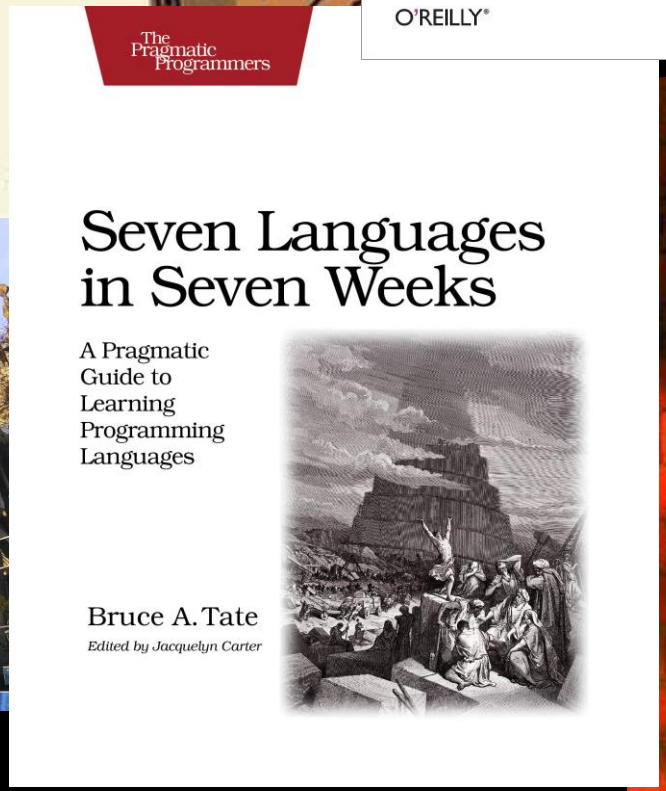


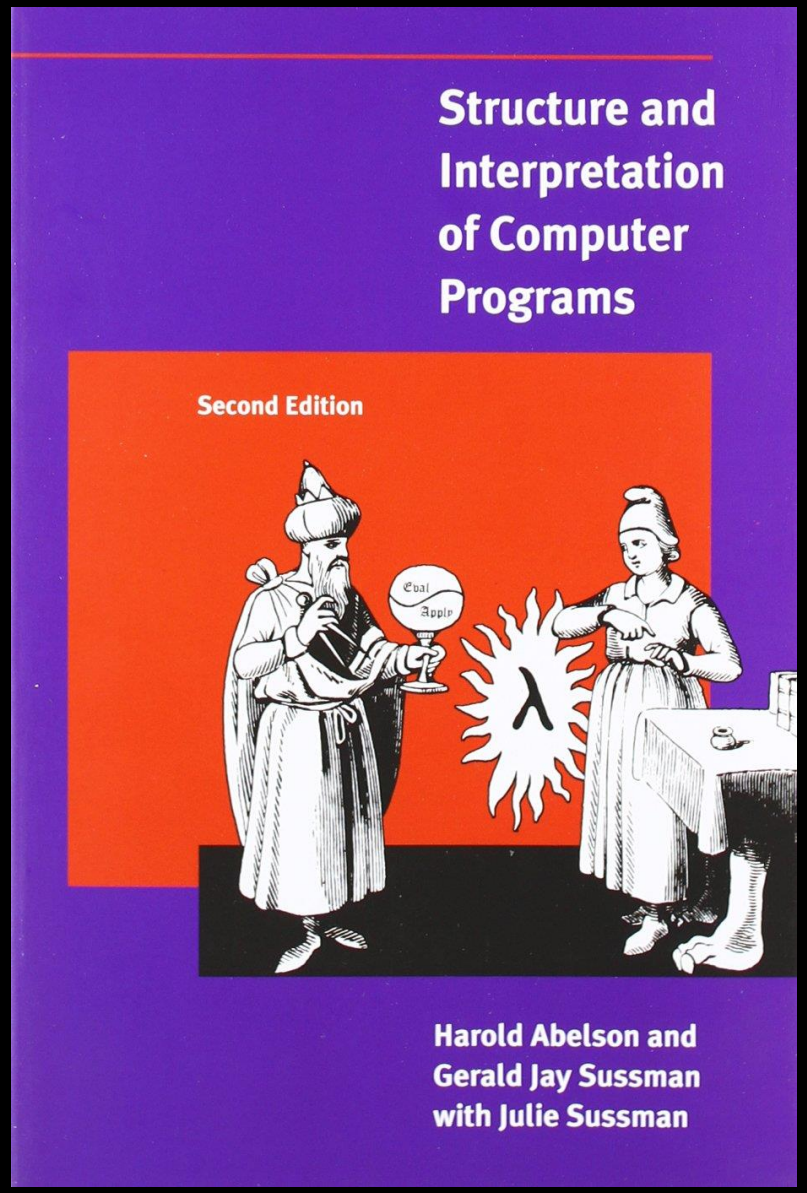












Structure and Interpretation of Computer Programs

Chaper 1.1

“I think that it’s extraordinarily important that we in computer science keep fun in computing.”

Alan J. Perlis

Dedication, SICP

Who is Alan. J Perlis?

- First recipient of the **Turing Award**
- Member of the team that developed (one of the most influential PLs ever)

ALGOL

- First president of the ACM
- Professor at Purdue, CMU and Yale

LUX ET VERITAS

acm

- Wrote "*Epigrams on Programming*"
  - "A LISP programmer knows the value of everything, but the cost of nothing."
  - "It is better to have 100 functions operate on one data structure than 10 functions on 10 data structures."

CARNEGIE MELLON UNIVERSITY
PITTSBURGH PENNSYLVANIA 1900

- *In Praise of APL: A Language for Lyrical Programming*

“A programmer should acquire good algorithms and idioms.”

Alan J. Perlis

Forward, SICP

“So you have all these algorithms at your disposal, learn them – that’s very important.”

Sean Parent

C++ Seasoning, 2013

“The discretionary exportable functionality entrusted to the individual Lisp programmer is more than an order of magnitude greater than that to be found within Pascal enterprises.”

Alan J. Perlis

Forward, SICP

"Lisp changes. The Scheme dialect used in this text has evolved from the original Lisp and differs from the latter in several important ways, including static scoping for variable binding and permitting functions to yield functions as values. In its semantic structure Scheme is as closely akin to Algol 60 as to early Lisps."

(1/3)

"Algol 60, never to be an active language again, lives on in the genes of Scheme and Pascal. It would be difficult to find two languages that are the communicating coin of two more different cultures than those gathered around these two languages."

(2/3)

"Pascal is for building pyramids – imposing, breathtaking, static structures built by armies pushing heavy blocks into place. Lisp is for building organisms - imposing, breathtaking, dynamic structures built by squads fitting fluctuating myriads of simpler organisms into place."
(3/3)

Alan J. Perlis
Forward, SICP

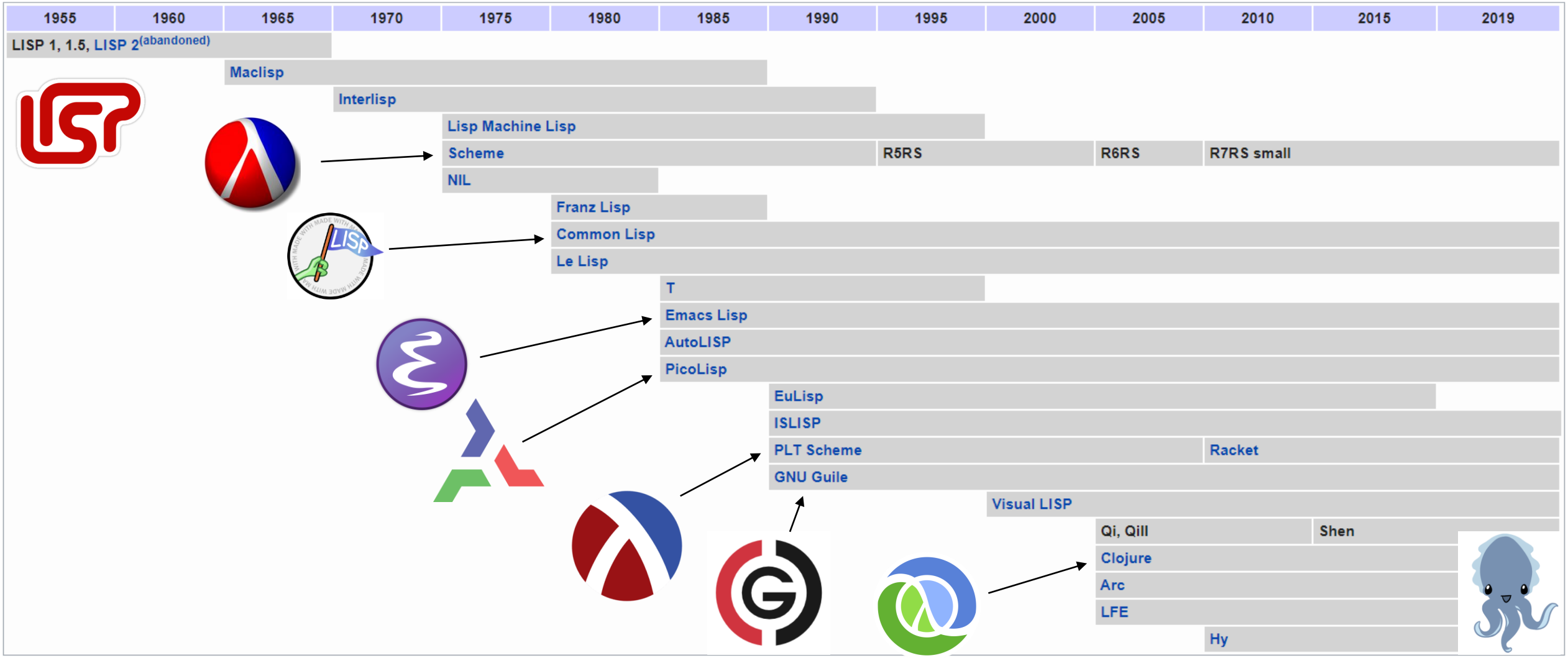
“Scheme, the dialect of Lisp that we use, is an attempt to bring together the power and elegance of Lisp and Algol.”

Preface to the First Edition, SICP

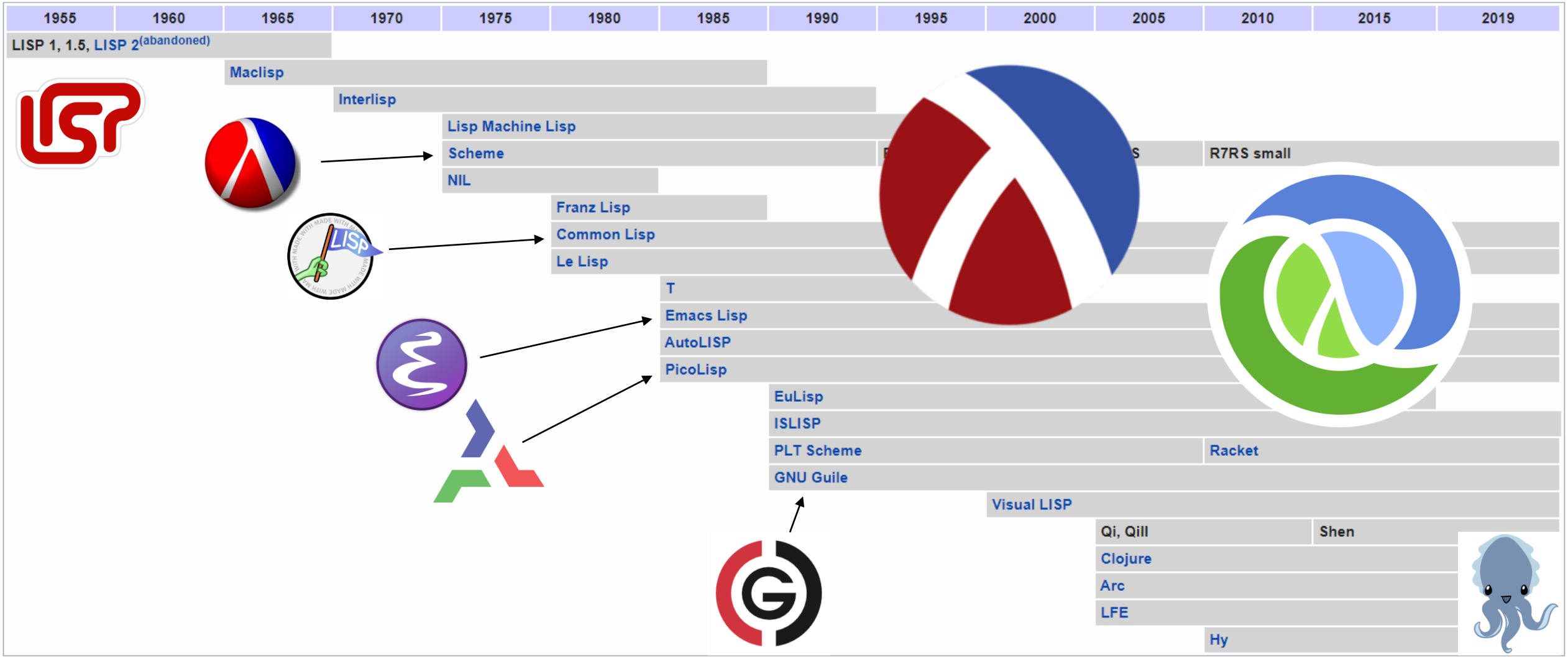
“The dialect of Lisp used in this book is called Scheme.”

Chapter 1, SICP

Timeline of Lisp dialects[edit]



Timeline of Lisp dialects[edit]



Late 50s

Mid 70s

90s

McCarthy

Scheme

Sussman & Steele

Racket
(originally PLT Scheme)

PLT Research Group

Algol Committee (incl. Perlis)

Racket (programming language) Common Lisp Scheme (programming language) Lisp (programming language) +1

How are Racket and Scheme different from each other?

Answer Follow · 5 Request 1

2 Answers

Shriram Krishnamurthi, Racketeer since the day it was born.
Answered September 19, 2016

Racket was originally a Scheme. We began a variant of Scheme, called PLT Scheme, because we wanted to have a quality open source implementation that interfaced well with graphical libraries, so that we could build the DrScheme programming environment.

As we built that environment and more things, we slowly realized a growing set of weaknesses in Scheme that prevented us from building better software. These led to a series of research innovations, all of which fed back into the language and into programs written in the language.

Eventually, we realized that we had grown a significantly bigger language on top of Scheme. It was proving problematic to call it a "Scheme" because this name was unfair both to Racket—making it look like a tiny language when it was a large one—and to the Scheme community, which may have felt the new language violated "Schemeness". Furthermore, even though the core of Racket is Scheme, it's not *exactly*: one of the changes was to make the language functional by default (specifically, **cons** produces immutable, not mutable, data). This last change was actually a non-backward-compatible change, and meant that many Scheme programs would fail to work as written.

It's difficult to explain more in the short span of a Quora post. I recommend looking at the article that explained the name change [From PLT Scheme to Racket] as well as our article that explains the identity of Racket (as different from other languages): The Racket Manifesto.

Quora Home 1 Answer Spaces Notifications 14 Search Qu

Racket (programming language) Common Lisp Scheme (program

Lisp (programming language) +1

How are Racket and Scheme different fro

Answer Follow · 5 Request

2 Answers

Shriram Krishnamurthi, Racketeer since the day it was

Answered September 19, 2016

Racket was originally a Scheme. We began a variant of Scheme because we wanted to have a quality open source implementa graphical libraries, so that we could build the DrScheme progra

As we built that environment and more things, we slowly realiz weaknesses in Scheme that prevented us from building better series of research innovations, all of which fed back into the lar written in the language.

Eventually, we realized that we had grown a significantly bigge Scheme. It was proving problematic to call it a "Scheme" becau both to Racket—making it look like a tiny language when it wa Scheme community, which may have felt the new language vic Furthermore, even though the core of Racket is Scheme, it's nc changes was to make the language functional by default (spec immutable, not mutable, data). This last change was actually a change, and meant that many Scheme programs would fail to

It's difficult to explain more in the short span of a Quora post. article that explained the name change [From PLT Scheme to R article that explains the identity of Racket (as different from ot Manifesto.

The Racket Manifesto

Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, Shriram Krishnamurthi, Eli Barzilay, Jay McCarthy, Sam Tobin-Hochstadt

Summit on Advances in Programming Languages, 2015

Abstract

The creation of a programming language calls for guiding principles that point the developers to goals. This article spells out the three basic principles behind the 20-year development of Racket. First, programming is about stating and solving problems, and this activity normally takes place in a context with its own language of discourse; good programmers ought to formulate this language as a programming language. Hence, *Racket is a programming language for creating new programming languages*. Second, by following this language-oriented approach to programming, systems become multi-lingual collections of interconnected components. Each language and component must be able to protect its specific invariants. In support, *Racket offers protection mechanisms to implement a full language spectrum*, from C-level bit manipulation to soundly typed extensions. Third, because Racket considers programming as problem solving in the correct language, *Racket also turns extra-linguistic mechanisms into linguistic constructs*, especially mechanisms for managing resources and projects. The paper explains these principles and how Racket lives up to them, presents the evaluation framework behind the design process, and concludes with a sketch of Racket's imperfections and opportunities for future improvements.

Comment

See also the upated, **journal version** of this paper.

Paper

PDF

POPL – Principles of Programming Languages

HOPL – History of Programming Languages

PLDI – Program Language Design and Implementation

SNAPL – Summit on Advances in Programming Languages

Organizing Committee HOPL IV

Richard P. Gabriel Co-chair
Dream Songs, Inc. & HPI

Guy L. Steele Jr. Co-chair
Oracle Labs
United States

"From Lisp we take the metalinguistic power that derives from the simple syntax, the uniform representation of programs as data objects, and the garbage-collected heap-allocated data. From Algol we take lexical scoping and block structure, which are gifts from the pioneers of programming-language design who were on the Algol committee."

Preface to the First Edition, SICP

“Lisp, whose name is an acronym for LISt Processing”

Chapter 1, SICP

“The thing to be multiplied is given a local name, *x* , which plays the same role that a pronoun plays in natural language.”

Chapter 1, SICP

“… the formal **parameters** are replaced by the actual **arguments** to which the procedure is applied”

Chapter 1, SICP

```
(+ 137 349)
486

(- 1000 334)
666

(* 5 99)
495

(/ 10 5)
2

(+ 2.7 10)
12.7
```

1.1.1 Expressions

```
(+ (* 3 (+ (* 2 4) (+ 3 5))) (+ (- 10 7) 6))

(+ (* 3
      (+ (* 2 4)
         (+ 3 5)))
   (+ (- 10 7)
   6))
```

1.1.1 Expressions

```
(define pi 3.14159)
(define radius 10)
(* pi (* radius radius))
314.159
(define circumference (* 2 pi radius))
circumference
62.8318
```

1.1.2 Naming and the Environment

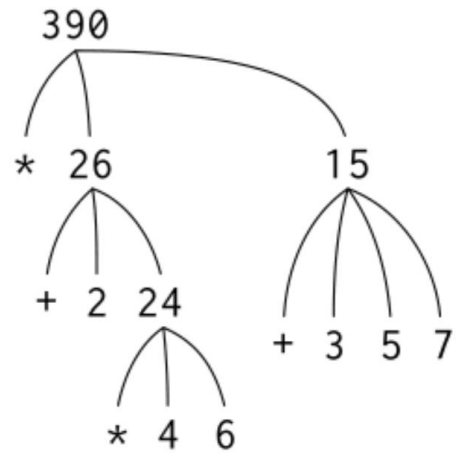


Figure 1.1: Tree representation, showing the value of each subcombination.

```
(* (+ 2 (* 4 6))
   (+ 3 5 7))
```

1.1.3 Evaluating Combinations

```
(define (square x) (* x x))

(define (sum-of-squares x y)
  (+ (square x) (square y)))

(sum-of-squares 3 4)
25
```

1.1.4 Compound Procedures

```
(f 5)
(sum-of-squares (+ a 1) (* a 2))

;; substitute

(sum-of-squares (+ 5 1) (* 5 2))
(+ (square 6) (square 10))
(+ (* 6 6) (* 10 10))
(+ 36 100)
136
```

1.1.5 The Substitution Model for Procedure Application

```
(f 5)
(sum-of-squares (+ a 1) (* a 2))

;; substitute

(sum-of-squares (+ 5 1) (* 5 2))
(+ (square 6) (square 10))
(+ (* 6 6) (* 10 10))
(+ 36 100)
136
```

Applicative order versus normal order

1.1.5 The Substitution Model for Procedure Application

```
(define (abs x)
  (cond ((> x 0) x)
        ((= x 0) 0)
        ((< x 0) (- x))))

(define (abs x)
  (if (< x 0)
      (- x)
      x))
```

1.1.6 Conditional Expressions and Predicates

Exercise 1.3: Define a procedure that takes three numbers as arguments and returns the sum of the squares of the two larger numbers.

```
(define (sum-square-two-largest lst)
  (~> lst
      (sort >)
      (take 2)
      (map (λ (x) (* x x)) _)
      (foldl + 0 _)))
```

1.1.6 Conditional Expressions and Predicates

Structure & Interpretation of Computer Programs

Harold Abelson

Gerald Jay Sussman

“magical language called Lisp”

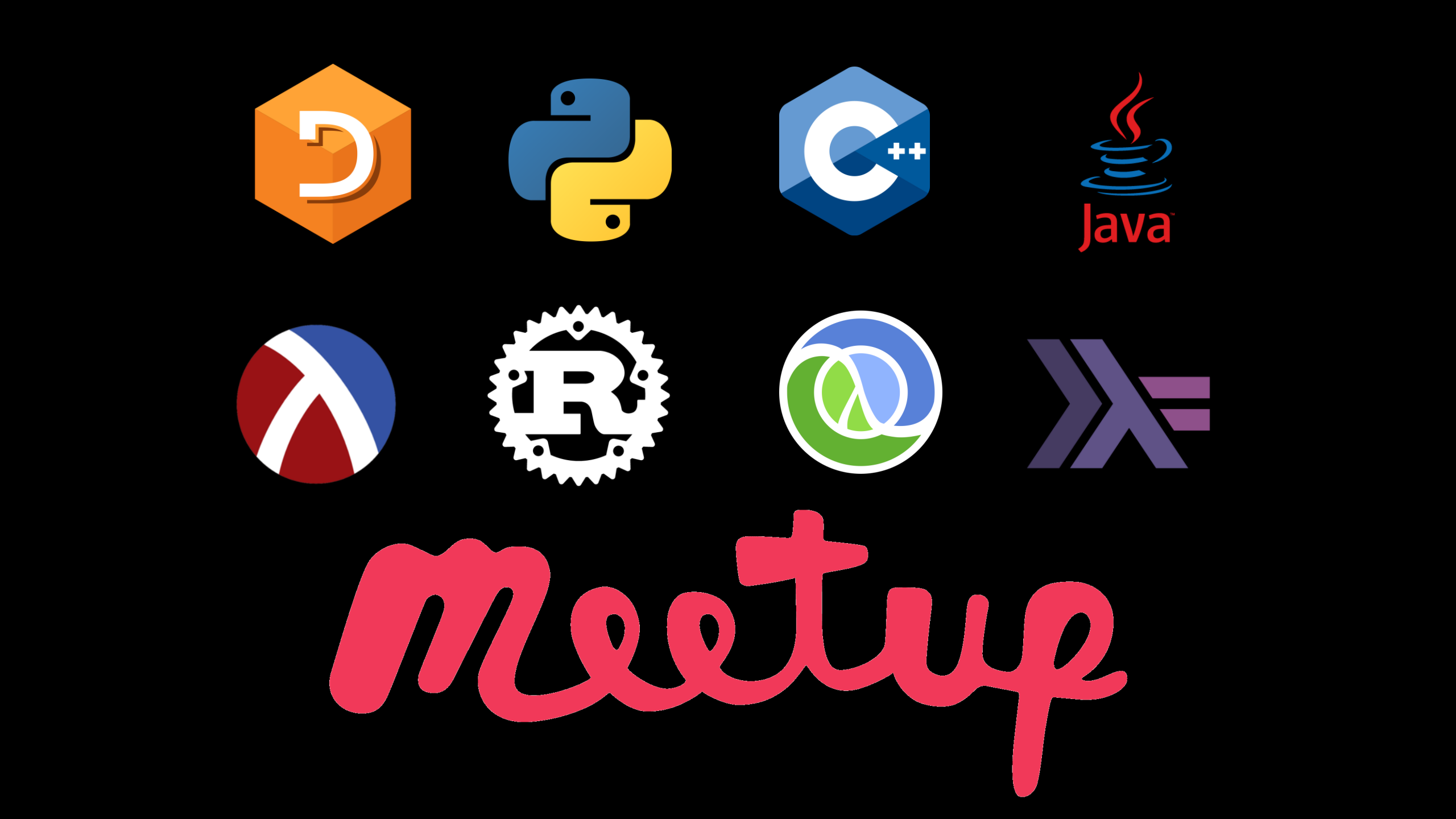
“become Master Programmers”

“the constraints imposed in building large software systems are the limitations of our own minds”

“you want to suppress detail”

Lecture 1: Overview and Introduction to Lisp

Lecture 1: Overview and Introduction to Lisp


Java
meetup