Lisp

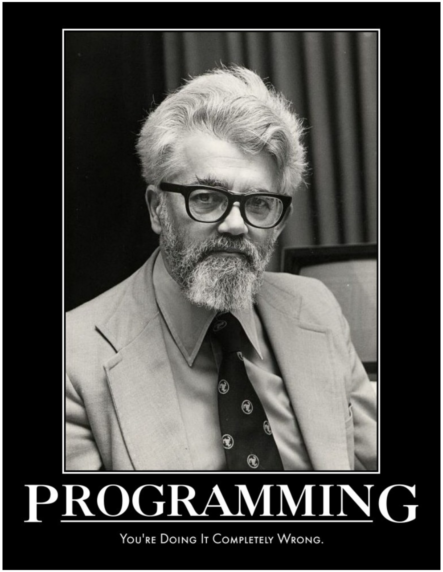
Moritz Heidkamp
moritz@twoticketsplease.de

August 20, 2011

Greenspun's Tenth Rule

Any sufficiently complicated C or Fortran program contains an ad hoc, informally-specified, bug-ridden, slow implementation of half of Common Lisp.

Where Does Lisp Come From?



John McCarthy, 1958

Where Does Lisp Come From?

LISP is now the second oldest programming language in present widespread use (after FORTRAN and not counting APT, which isn't used for programming per se).

John McCarthy: History of Lisp, 1979.

Where Does Lisp Come From?

IBM 704, 1960/61

A Lisp Program From ca. 1961

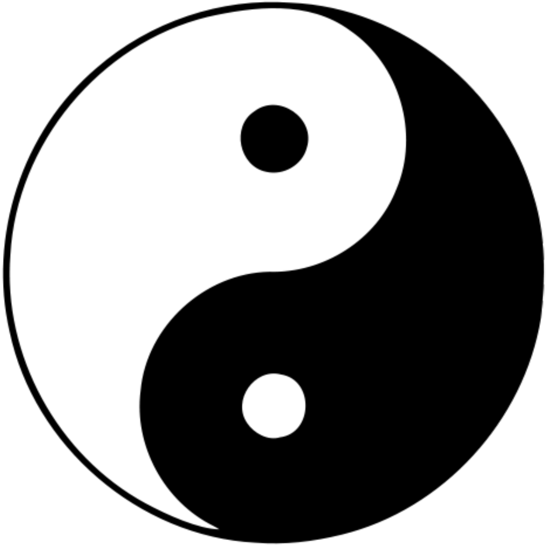
```
DEFINE ((
(MEMBER (LAMBDA (A X) (COND ((NULL X) F)
      ((EQ A (CAR X)) T) (T (MEMBER A (CDR X))) )))
(UNION (LAMBDA (X Y) (COND ((NULL X) Y) ((MEMBER
      (CAR X) Y) (UNION (CDR X) Y)) (T (CONS (CAR X)
      (UNION (CDR X) Y))) )))
(INTERSECTION (LAMBDA (X Y) (COND ((NULL X) NIL)
      ((MEMBER (CAR X) Y) (CONS (CAR X) (INTERSECTION
      (CDR X) Y))) (T (INTERSECTION (CDR X) Y)) )))
))
INTERSECTION ((A1 A2 A3) (A1 A3 A5))
UNION ((X Y Z) (U V W X))
```

What Is Lisp?

λ

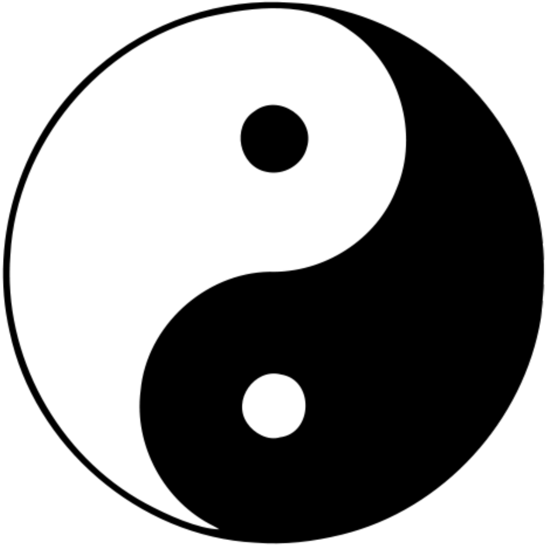
Functional

What Is Lisp?



Homoiconic
Code = Data

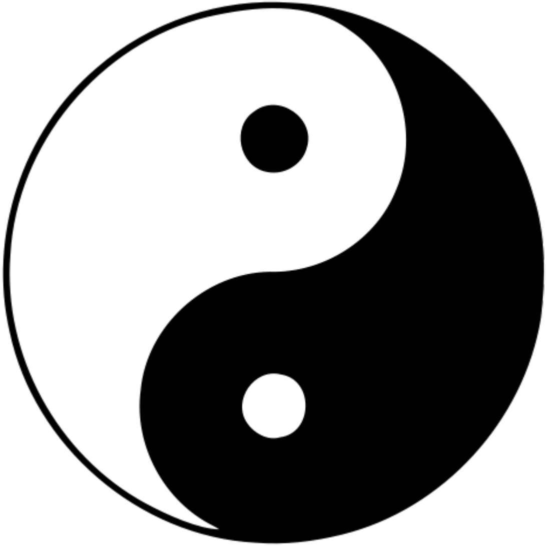
What Is Lisp?



Homoiconic

Code = Data

What Is Lisp?

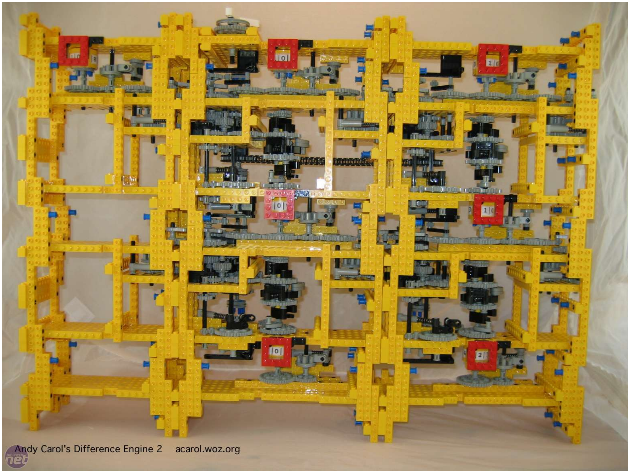


Homoiconic
Code = Data

What Is Lisp?

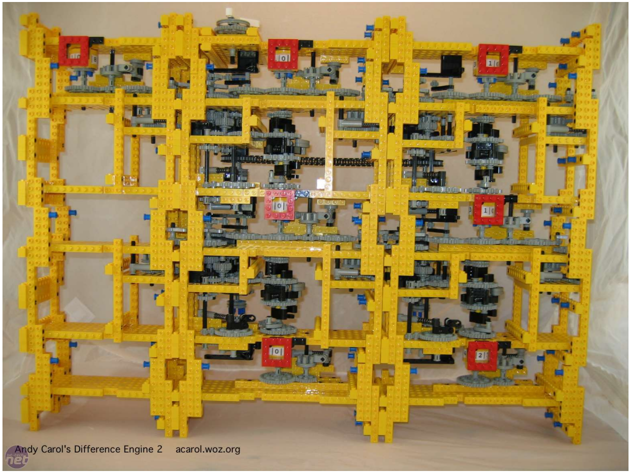
Garbage Collection

What Is Lisp?



Programmable Programming Language
Multi-paradigm

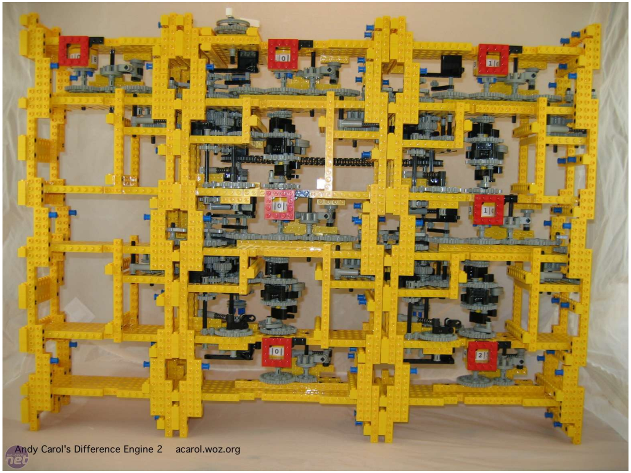
What Is Lisp?



Programmable Programming Language

Multi-paradigm

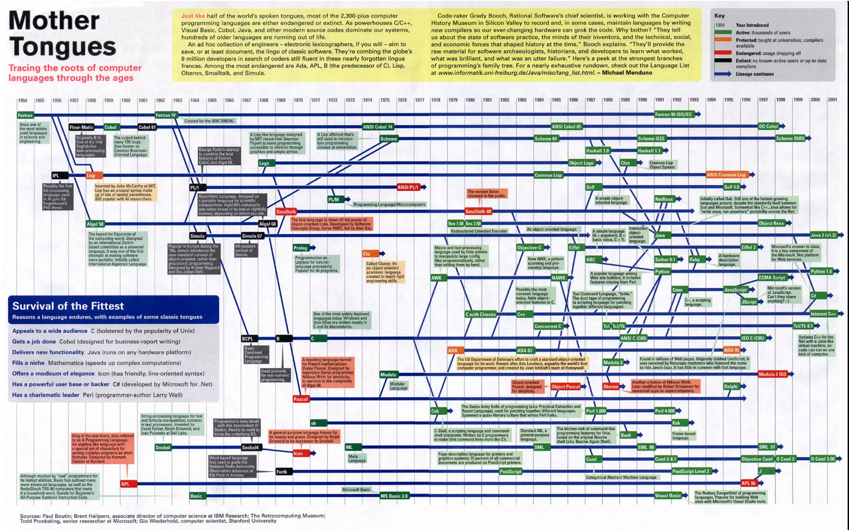
WHAT IS LISP?



Programmable Programming Language
Multi-paradigm

What Is Lisp?

Dialects



Mother Tongues

Tracing the roots of computer languages through the ages

Just like half of the world's spoken tongues, most of the 2,300-plus computer programming languages are either endangered or extinct. As powerhouses C/C++, Visual Basic, Cobol, Java, and other modern source codes dominate our systems, hundreds of older languages are running out of life.

An ad hoc collection of engineers – electronic lexicographers, if you will – aim to save, or at least document, the lingo of classic software. They're combing the globe's 9 million developers in search of coders still fluent in these nearly forgotten lingua francas. Among the most endangered are Ada, APL, B (the predecessor of C), Lisp, Oberon, Smalltalk, and Simula.

Code-raker Grady Booch, Rational Software's chief scientist, is working with the Computer History Museum in Silicon Valley to record and, in some cases, maintain languages by writing new compilers so our ever-changing hardware can grok the code. Why bother? "They tell us about the state of software practice, the minds of their inventors, and the technical, social, and economic forces that shaped history at the time," Booch explains. "They'll provide the raw material for software archaeologists, historians, and developers to learn what worked, what was brilliant, and what was an utter failure." Here's a peek at the strongest branches of programming's family tree. For a nearly exhaustive rundown, check out the Language List at *www.informatik.uni-freiburg.de/Java/misc/lang_list.html.* – **Michael Menduno**

Key

1954	**Year Introduced**
🟩	**Active:** thousands of users
🟧	**Protected:** taught at universities; compilers available
🟥	**Endangered:** usage dropping off
⬛	**Extinct:** no known active users or up-to-date compilers
➡	**Lineage continues**

Fortran: Once one of the most widely used languages in science and engineering.

Fortran IV: Created for the IBM 7090/94.

Flow-Matic: Originally B-0. One of the first English-like data-processing languages.

Cobol: The culprit behind many Y2K bugs. Also known as Common Business-Oriented Language.

IPL: Possibly the first list-processing language; used in AI guru Ed Feigenbaum's PhD thesis.

Lisp: Invented by John McCarthy at MIT, Lisp has an unusual syntax made up of lots of nested parentheses. Still popular with AI researchers.

Algol 58: The hoped-for Esperanto of the computing world. Designed by an international Zurich-based committee as a universal language, it was one of the first attempts at making software more portable. Initially called International Algebraic Language.

PL/1: George Radin's attempt to combine the best features of Fortran, Cobol, and Algol 60.

Algol 68: Algorithmic Language, designed as a portable language for scientific computations. Algol 68's complexity was either ahead of its time or rightfully doomed, depending on whom you ask.

Simula: Popular in Europe during the '70s, Simula introduced the now-standard concept of object-oriented, rather than procedural, programming. Designed by Kristen Nygaard and Ole-Johan Dahl.

Simula 67: All-purpose version of Simula.

Logo: A Lisp-like language designed by MIT researcher Seymour Papert to make programming accessible to children through graphics and simple syntax.

Scheme: A Lisp offshoot that's still used in introductory programming classes at universities.

PL/M: Programming Language/Microcomputers

Smalltalk: The first language to show off the power of object-oriented code. Developed by Software Concepts Group, Xerox PARC, led by Alan Kay.

Smalltalk-80: The version Xerox released to the public.

Prolog: Programmation en Logique for natural-language processing. Popular for AI programs.

Clu: Called Cluster, it's an object-oriented academic language created to teach rigid engineering skills.

Rex 1.00: Restructured Extended Executor

AWK: Macro and text-processing language used by Unix admins to manipulate large config files programmatically, rather than editing them by hand.

NAWK: New AWK, a pattern scanning and processing language.

Objective-C: An object-oriented language.

Self: A simple object-oriented language.

ABC: A simple language (A = argument, B = basic value, C = ?).

Sather 0.1: Interactive object-oriented language.

Python: A popular language among Web site builders, it includes features missing from Perl.

Clos: Common Lisp Object System

Java: Initially called Oak. Still one of the fastest-growing languages around, despite the standards feud between Sun and Microsoft. Somewhat like C++, Java allows for "write once, run anywhere" portability across the Net.

Ruby: A hardware description language.

C#: Microsoft's answer to Java, it is a key component of the Microsoft .Net platform for Web services.

Cmm: C--, a scripting language.

JScript: Microsoft's version of JavaScript. Can't they share anything? :-)

C++: Possibly the most common language today. Adds object-oriented features to C.

Tcl: Tool Command Language, "tickle." The duct tape of programming (a scripting language for patching together different languages).

Internet C++: Updates C++ for the Net with a Java-like virtual machine, so code can run on any kind of computer.

C: One of the most widely deployed languages today. Windows and Unix OSes are written mostly in C and its descendants.

BCPL: Basic Combined Programming Language

B: Used primarily for non-numeric programming.

Pascal: A teaching language named for French mathematician Blaise Pascal. Designed by legendary Swiss programmer Niklaus Wirth for simplicity, in reaction to the complexity of Algol 68.

Modula: Modular Language

ADA: The US Department of Defense's effort to craft a standard object-oriented language for its work. Named after Ada Lovelace, arguably the world's first computer programmer, and created by Jean Ichbiah's team at Honeywell.

Object Pascal: Object-oriented Pascal, designed for simplicity.

Oberon: Another creation of Niklaus Wirth. Later modified by Robert Griesemer for numerical apps on supercomputers.

JavaScript: Found in millions of Web pages. Originally dubbed LiveScript, it was renamed by Netscape marketers who licensed the name to ride Java's buzz. It has little in common with that language.

Perl 1.000: The Swiss Army Knife of programming (aka Practical Extraction and Report Language), used for patching together different languages. Spawned a quasi-literary culture that writes Perl haiku.

Csh: C-Shell, a scripting language and command-shell interpreter. Written by C programmers to make Unix command lines more like C's.

SML: Standard ML, a general-purpose language.

Bash: The kitchen sink of command-line programming features for Unix, based on the original Bourne shell (aka Bourne Again Shell).

Ksh: Frame-based language.

Snobol: String-processing language for text and formula manipulation, common in text processors. Invented by David Farber, Ralph Griswold, and Ivan Polonsky at Bell Labs.

Snobol4: Programmer's easy street – with this descendant of Snobol, there's no need to know the underlying OS.

Icon: A general-purpose language known for its beauty and grace. Designed by Ralph Griswold to be successor to Snobol4.

ML: Meta Language

PostScript: Page-description language for printers and graphics systems; 75 percent of all commercial documents are produced on PostScript printers.

Caml: Categorical Abstract Machine Language

APL: King of the one-liners, also referred to as A Programming Language. An algebra-like language with a special set of characters for writing complex programs as short formulas. Designed by Kenneth Iverson at Harvard.

Forth: Word-based language first used to guide the National Radio Astronomy Observatory telescope at Kitt Peak in Arizona.

Basic: Although mocked by "real" programmers for its limited abilities, Basic has outlived many more advanced languages, as well as the RadioShack TRS-80 computers that made it a household word. Stands for Beginner's All-Purpose Symbolic Instruction Code.

MS Basic 2.0: Microsoft Basic

Visual Basic: The Rodney Dangerfield of programming languages. Popular for building Web sites with Microsoft's Visual Studio tools.

Survival of the Fittest

Reasons a language endures, with examples of some classic tongues

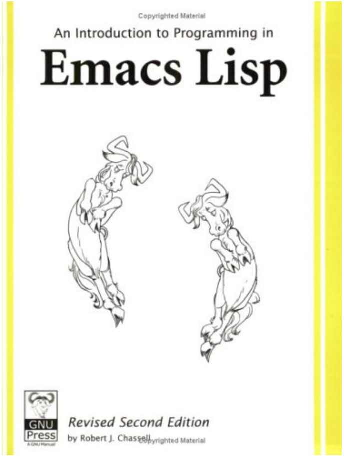
Appeals to a wide audience	C (bolstered by the popularity of Unix)
Gets a job done	Cobol (designed for business-report writing)
Delivers new functionality	Java (runs on any hardware platform)
Fills a niche	Mathematica (speeds up complex computations)
Offers a modicum of elegance	Icon (has friendly, line-oriented syntax)
Has a powerful user base or backer	C# (developed by Microsoft for .Net)
Has a charismatic leader	Perl (programmer-author Larry Wall)

Sources: Paul Boutin; Brent Hailpern, associate director of computer science at IBM Research; The Retrocomputing Museum; Todd Proebsting, senior researcher at Microsoft; Gio Wiederhold, computer scientist, Stanford University

Common Lisp

1984

Emacs Lisp


Copyrighted Material

An Introduction to Programming in

Emacs Lisp

GNU Press

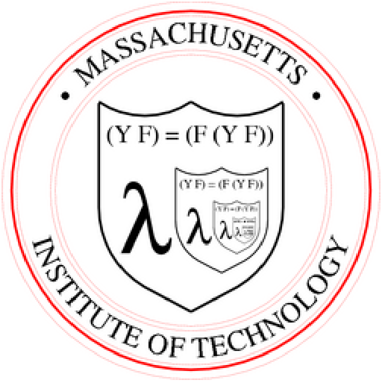
Revised Second Edition

by Robert J. Chassell

Copyrighted Material

1984

Scheme



1975

Clojure

2007

S-Expressions: Atoms

```
these are 4 atoms

null? call/cc set!

"hello, world"

+ 3.5 - foo@bar
```

S-Expressions: Atoms

```
these are 4 atoms

null? call/cc set!

"hello, world"

+ 3.5 - foo@bar
```

S-Expressions: Atoms

```
these are 4 atoms

null? call/cc set!

"hello, world"

+ 3.5 - foo@bar
```

S-Expressions: Atoms

```
these are 4 atoms
```

```
null? call/cc set!
```

```
"hello, world"
```

```
+ 3.5 - foo@bar
```

S-Expressions: Atoms

```
these are 4 atoms

null? call/cc set!

"hello, world"

+ 3.5 - foo@bar
```

S-Expressions: Pairs, Cons Cells, Lists

```
() nil

(1 . 5) (<car> . <cdr>)
(<first> . <rest>)

(1 . ())  = (1)

(1 . (2 . (3 . ()))) = (1 2 3)

(a list of 5 elements)

((nested (list)) (with . pair))
```

S-Expressions: Pairs, Cons Cells, Lists

```
() nil

(1 . 5) (<car> . <cdr>)
(<first> . <rest>)

(1 . ())  = (1)

(1 . (2 . (3 . ()))) = (1 2 3)

(a list of 5 elements)

((nested (list)) (with . pair))
```

S-Expressions: Pairs, Cons Cells, Lists

```
() nil

(1 . 5) (<car> . <cdr>)
(<first> . <rest>)

(1 . ()) = (1)

(1 . (2 . (3 . ()))) = (1 2 3)

(a list of 5 elements)

((nested (list)) (with . pair))
```

S-Expressions: Pairs, Cons Cells, Lists

```
() nil

(1 . 5) (<car> . <cdr>)
(<first> . <rest>)

(1 . ())  = (1)

(1 . (2 . (3 . ()))) = (1 2 3)

(a list of 5 elements)

((nested (list)) (with . pair))
```

S-Expressions: Pairs, Cons Cells, Lists

```
() nil

(1 . 5) (<car> . <cdr>)
(<first> . <rest>)

(1 . ()) = (1)

(1 . (2 . (3 . ()))) = (1 2 3)

(a list of 5 elements)

((nested (list)) (with . pair))
```

S-Expressions: Pairs, Cons Cells, Lists

```
() nil

(1 . 5) (<car> . <cdr>)
(<first> . <rest>)

(1 . ()) = (1)

(1 . (2 . (3 . ()))) = (1 2 3)

(a list of 5 elements)

((nested (list)) (with . pair))
```

S-Expressions: Pairs, Cons Cells, Lists

```
() nil

(1 . 5) (<car> . <cdr>)
(<first> . <rest>)

(1 . ()) = (1)

(1 . (2 . (3 . ()))) = (1 2 3)

(a list of 5 elements)

((nested (list)) (with . pair))
```

S-Expressions: Pairs, Cons Cells, Lists

```
() nil

(1 . 5) (<car> . <cdr>)
(<first> . <rest>)

(1 . ()) = (1)

(1 . (2 . (3 . ()))) = (1 2 3)

(a list of 5 elements)

((nested (list)) (with . pair))
```

S-Expressions: Pairs, Cons Cells, Lists

```
() nil

(1 . 5) (<car> . <cdr>)
(<first> . <rest>)

(1 . ()) = (1)

(1 . (2 . (3 . ()))) = (1 2 3)

(a list of 5 elements)

((nested (list)) (with . pair))
```

S-Expressions: Pairs, Cons Cells, Lists

```
() nil

(1 . 5) (<car> . <cdr>)
(<first> . <rest>)

(1 . ())  = (1)

(1 . (2 . (3 . ()))) = (1 2 3)

(a list of 5 elements)

((nested (list)) (with . pair))
```

S-Expressions: Pairs, Cons Cells, Lists

```
() nil

(1 . 5) (<car> . <cdr>)
(<first> . <rest>)

(1 . ())  = (1)

(1 . (2 . (3 . ()))) = (1 2 3)

(a list of 5 elements)

((nested (list)) (with . pair))
```

S-Expressions: Pairs, Cons Cells, Lists

```
() nil

(1 . 5) (<car> . <cdr>)
(<first> . <rest>)

(1 . ())  = (1)

(1 . (2 . (3 . ()))) = (1 2 3)

(a list of 5 elements)

((nested (list)) (with . pair))
```

Semantics

```
(operator operand operand ...)
```

```
(+ 12.5 (/ 93.8 20 (+ 10 (* 10 67))))
```

```
(car (list 12.5 "hey"))
```

Semantics

```
(operator operand operand ...)
```

```
(+ 12.5 (/ 93.8 20 (+ 10 (* 10 67))))
```

```
(car (list 12.5 "hey"))
```

Semantics

```
(operator operand operand ...)

(+ 12.5 (/ 93.8 20 (+ 10 (* 10 67))))

(car (list 12.5 "hey"))
```

Special Forms

Binding: `(define lotto-gewinn (* 1000 1000))`

Functions: `(lambda (x y) (/ (* x x) y))`

Quoting: `(quote (foo bar)) = '(foo bar)`

Conditionals:

```
(cond ((< jackpot 100)
        (display "whatever"))
       ((> jackpot 100000)
        (display "not bad!"))
       (else
        (display "could be worse")))
```

Special Forms

Binding: `(define lotto-gewinn (* 1000 1000))`

Functions: `(lambda (x y) (/ (* x x) y))`

Quoting: `(quote (foo bar)) = '(foo bar)`

Conditionals:

```
(cond ((< jackpot 100)
        (display "whatever"))
       ((> jackpot 100000)
        (display "not bad!"))
       (else
        (display "could be worse")))
```

Special Forms

Binding: `(define lotto-gewinn (* 1000 1000))`

Functions: `(lambda (x y) (/ (* x x) y))`

Quoting: `(quote (foo bar)) = '(foo bar)`

Conditionals:

```
(cond ((< jackpot 100)
        (display "whatever"))
       ((> jackpot 100000)
        (display "not bad!"))
       (else
        (display "could be worse")))
```

Special Forms

Binding: `(define lotto-gewinn (* 1000 1000))`

Functions: `(lambda (x y) (/ (* x x) y))`

Quoting: `(quote (foo bar)) = '(foo bar)`

Conditionals:

```
(cond ((< jackpot 100)
        (display "whatever"))
       ((> jackpot 100000)
        (display "not bad!"))
       (else
        (display "could be worse")))
```

Some functions

```
(cons a b) => (a . b)
```

```
(car '(1 2 3)) => 1
```

```
(cdr '(1 2 3)) => (2 3)
```

```
(eq? a b) => #t / #f
```

Some functions

```
(cons a b) => (a . b)
```

```
(car '(1 2 3)) => 1
```

```
(cdr '(1 2 3)) => (2 3)
```

```
(eq? a b) => #t / #f
```

Some functions

```
(cons a b) => (a . b)
```

```
(car '(1 2 3)) => 1
```

```
(cdr '(1 2 3)) => (2 3)
```

```
(eq? a b) => #t / #f
```

Some functions

```
(cons a b) => (a . b)
```

```
(car '(1 2 3)) => 1
```

```
(cdr '(1 2 3)) => (2 3)
```

```
(eq? a b) => #t / #f
```

Defining Functions

Functions are values!

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

```
(square 11) => 121
```

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

```
((lambda (x) (* x x)) 11) => 121
```

Defining Functions

Functions are values!

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

```
(square 11) => 121
```

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

```
((lambda (x) (* x x)) 11) => 121
```

Defining Functions

Functions are values!

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

```
(square 11) => 121
```

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

```
((lambda (x) (* x x)) 11) => 121
```

Defining Functions

Functions are values!

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

```
(square 11) => 121
```

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

```
((lambda (x) (* x x)) 11) => 121
```

Defining Functions

Functions are values!

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

```
(square 11) => 121
```

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

```
((lambda (x) (* x x)) 11) => 121
```

Defining Functions

Functions are values!

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

```
(square 11) => 121
```

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

```
((lambda (x) (* x x)) 11) => 121
```

Higher Order Functions

Functions which take functions as arguments and/or return functions.

```
(map square '(3 4 5)) => (9 16 25)
```

```
(for-each (lambda (x y)
            (display (format "~A ~A" x y))
            (newline))
          '(99 1001 42)
          '(red-balloons nights wtf/s))
```

```
99 red-balloons
1001 nights
42 wtf/s
```

Higher Order Functions

Functions which take functions as arguments and/or return functions.

```
(map square '(3 4 5)) => (9 16 25)
```

```
(for-each (lambda (x y)
            (display (format "~A ~A" x y))
            (newline))
          '(99 1001 42)
          '(red-balloons nights wtf/s))
```

```
99 red-balloons
1001 nights
42 wtf/s
```

Higher Order Functions

Functions which take functions as arguments and/or return functions.

```
(map square '(3 4 5)) => (9 16 25)
```

```
(for-each (lambda (x y)
            (display (format "~A ~A" x y))
            (newline))
          '(99 1001 42)
          '(red-balloons nights wtf/s))
```

```
99 red-balloons
1001 nights
42 wtf/s
```

Higher Order Functions

Functions which take functions as arguments and/or return functions.

```
(map square '(3 4 5)) => (9 16 25)
```

```
(for-each (lambda (x y)
            (display (format "~A ~A" x y))
            (newline))
          '(99 1001 42)
          '(red-balloons nights wtf/s))
```

```
99 red-balloons
1001 nights
42 wtf/s
```

Macros

```
(unless (too-late?)
  (do-taxes)
  (go-shopping))
```

```
(cond ((not (too-late?))
       (do-taxes)
       (go-shopping)))
```

```
(define-syntax unless
  (syntax-rules ()
    ((unless condition exp ...)
     (cond ((not condition) exp ...)))))
```

Macros

```
(unless (too-late?)
  (do-taxes)
  (go-shopping))

(cond ((not (too-late?))
       (do-taxes)
       (go-shopping)))

(define-syntax unless
  (syntax-rules ()
    ((unless condition exp ...)
     (cond ((not condition) exp ...)))))
```

Macros

```
(unless (too-late?)
  (do-taxes)
  (go-shopping))

(cond ((not (too-late?))
       (do-taxes)
       (go-shopping)))

(define-syntax unless
  (syntax-rules ()
    ((unless condition exp ...)
     (cond ((not condition) exp ...)))))
```

The World Seen Through The Eyes Of A Lisper

Arithmetische Ausdrücke

```
39 - 210 / (3 + 10 * 67)
```

```
(- 39 (/ 210 (+ 3 (* 10 67))))
```

The World Seen Through The Eyes Of A Lisper

Arithmetische Ausdrücke

```
39 - 210 / (3 + 10 * 67)
```

```
(- 39 (/ 210 (+ 3 (* 10 67))))
```

The World Seen Through The Eyes Of A Lisper

XML

```
<html>
  <head>
    <title>Lisp &gt; all</title>
  </head>
  <body>
    <h1>Welcome!</h1>
    <p>Here you will find:</p>
    <ul>
      <li><a href="/lispy-times">Lispy Times!</a></li>
      <li><a href="/other-stuff">Some other stuff</a></li>
      <li><a href="/more">and MORE</a></li>
    </ul>
  </body>
</html>
```

SXML

```
(html
 (head
  (title "Lisp > all"))
 (body (h1 "Welcome!")
       (p "Here you will find:")
       (ul
        (li (a (@ (href "/lispy-times")) "Lispy Times!"))
        (li (a (@ (href "/other-stuff")) "Some other stuff"))
        (li (a (@ (href "/more")) "and MORE")))))
```

The World Seen Through The Eyes Of A Lisper

XML

```
<html>
  <head>
    <title>Lisp &gt; all</title>
  </head>
  <body>
    <h1>Welcome!</h1>
    <p>Here you will find:</p>
    <ul>
      <li><a href="/lispy-times">Lispy Times!</a></li>
      <li><a href="/other-stuff">Some other stuff</a></li>
      <li><a href="/more">and MORE</a></li>
    </ul>
  </body>
</html>
```

SXML

```
(html
 (head
  (title "Lisp > all"))
 (body (h1 "Welcome!")
       (p "Here you will find:")
       (ul
        (li (a (@ (href "/lispy-times")) "Lispy Times!"))
        (li (a (@ (href "/other-stuff")) "Some other stuff"))
        (li (a (@ (href "/more")) "and MORE")))))
```

The World Seen Through The Eyes Of A Lisper

Regular Expressions

```
/^(foo|bar|baz)\s+\d+/
```

SREs

```
(seq bol (submatch (or "foo" "bar" "baz"))
 (+ space) (+ number))
```

The World Seen Through The Eyes Of A Lisper

Regular Expressions

```
/^(foo|bar|baz)\s+\d+/
```

SREs

```
(seq bol (submatch (or "foo" "bar" "baz"))
 (+ space) (+ number))
```

The World Seen Through The Eyes Of A Lisper

SQL

```
SELECT firstname, lastname, company
FROM members AS m
LEFT JOIN interests AS i ON i.member_id = m.id
WHERE age > 18
ORDER BY lastname, firstname;
```

SSQL

```
(select (columns firstname lastname company)
  (from (join left (as members m) (as interests i)
              (on (= (col m id) (col i member_id)))))
  (where (> age 19))
  (order (desc lastname) firstname))
```

The World Seen Through The Eyes Of A Lisper

SQL

```
SELECT firstname, lastname, company
FROM members AS m
LEFT JOIN interests AS i ON i.member_id = m.id
WHERE age > 18
ORDER BY lastname, firstname;
```

SSQL

```
(select (columns firstname lastname company)
  (from (join left (as members m) (as interests i)
              (on (= (col m id) (col i member_id)))))
  (where (> age 19))
  (order (desc lastname) firstname))
```

Why Lisp?

So why would you use that?

Paul Graham: Beating The Averages
http://www.paulgraham.com/avg.html

Why Lisp?

So why would you use that?

Paul Graham: Beating The Averages
`http://www.paulgraham.com/avg.html`

Chicken Scheme

CHICKENscheme
A PRACTICAL AND PORTABLE SCHEME SYSTEM

`http://www.call-cc.org/`

- Compiler translates Scheme to C
- Interpreter, mixing is possible
- portable and embeddable
- more than 400 extensions, “eggs”
- `chicken-install postgresql`

Chicken Scheme

CHICKENscheme
A PRACTICAL AND PORTABLE SCHEME SYSTEM

`http://www.call-cc.org/`

- Compiler translates Scheme to C
- Interpreter, mixing is possible
- portable and embeddable
- more than 400 extensions, “eggs”
- `chicken-install postgresql`

Chicken Scheme

CHICKENscheme

A PRACTICAL AND PORTABLE SCHEME SYSTEM

`http://www.call-cc.org/`

- Compiler translates Scheme to C
- Interpreter, mixing is possible
- portable and embeddable
- more than 400 extensions, “eggs”
- `chicken-install postgresql`

Chicken Scheme

CHICKENscheme
A PRACTICAL AND PORTABLE SCHEME SYSTEM

`http://www.call-cc.org/`

- Compiler translates Scheme to C
- Interpreter, mixing is possible
- portable and embeddable
- more than 400 extensions, “eggs”
- `chicken-install postgresql`

Chicken Scheme

CHICKENscheme

A PRACTICAL AND PORTABLE SCHEME SYSTEM

`http://www.call-cc.org/`

- Compiler translates Scheme to C
- Interpreter, mixing is possible
- portable and embeddable
- more than 400 extensions, "eggs"
- `chicken-install postgresql`

Chicken Scheme

CHICKENscheme
A PRACTICAL AND PORTABLE SCHEME SYSTEM

`http://www.call-cc.org/`

- Compiler translates Scheme to C
- Interpreter, mixing is possible
- portable and embeddable
- more than 400 extensions, “eggs”
- `chicken-install postgresql`

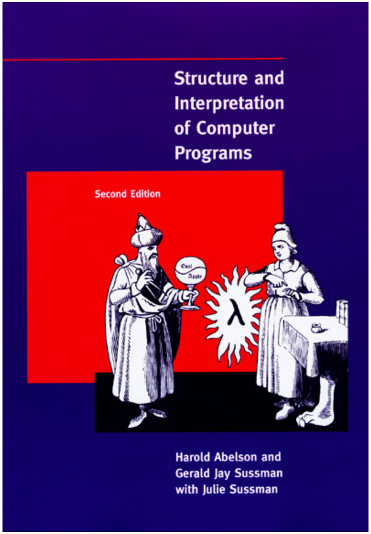
Chicken Scheme

CHICKENscheme
A PRACTICAL AND PORTABLE SCHEME SYSTEM

`http://www.call-cc.org/`

- Compiler translates Scheme to C
- Interpreter, mixing is possible
- portable and embeddable
- more than 400 extensions, “eggs”
- `chicken-install postgresql`

Recommended Reading

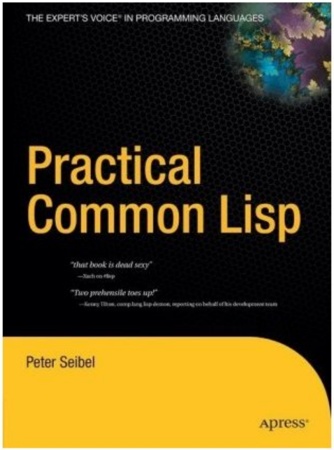

Structure and Interpretation of Computer Programs

Second Edition

Harold Abelson and Gerald Jay Sussman with Julie Sussman

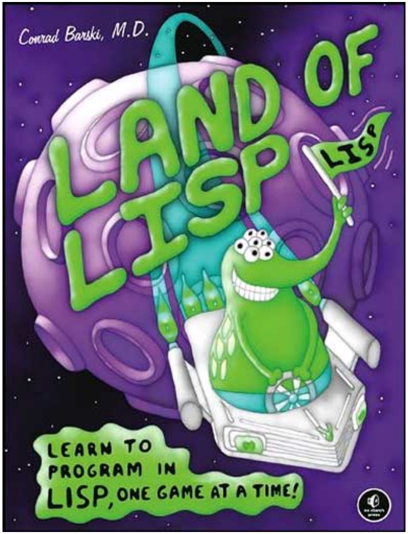
http://mitpress.mit.edu/sicp/full-text/book/book.html

Recommended Reading

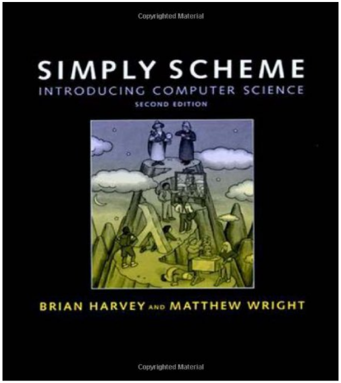

THE EXPERT'S VOICE® IN PROGRAMMING LANGUAGES
Practical Common Lisp
"that book is dead sexy"
"Two prehensile toes up!"
Peter Seibel
Apress®

http://www.gigamonkeys.com/book/

Recommended Reading

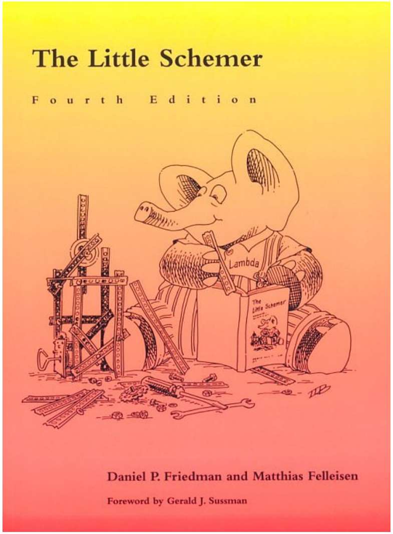


Recommended Reading



http://www.cs.berkeley.edu/~bh/simply-toc.html

Recommended Reading



The Little Schemer

Fourth Edition

Daniel P. Friedman and Matthias Felleisen

Foreword by Gerald J. Sussman

References

John McCarthy: History of Lisp, 1979
`http://www-formal.stanford.edu/jmc/history/lisp/lisp.html`

Association of Lisp Users `http://lisp.org/`

Scheme `http://schemers.org/`
`http://schemewiki.org/`

Common Lisp `http://www.cliki.net/`