PDF pour 18216 scan

Le présent fichier PDF **Sommaire & introduction** concerne le document

Lisp, a gentle introduction to symbolic computation

David S. Touretzky est l'auteur de cet ouvrage de 384 pages, édité en 1985, en anglais, par Harper & Row, New York.

Cet ouvrage est disponible sous le numéro **AC 18216** dans l'inventaire de la collection de l'ACONIT, Association pour un conservatoire de l'informatique et de la télématique, Grenoble.

Auparavant il appartenait aux collections de la Bibliothèque Municipale de Grenoble, d'où la référence MA 2001 007 0003 qui apparaît sur la page de couverture.

Le présent PDF fournit un scan d'extraits de cet ouvrage, avec copie de :

-la page de couverture,
-la 4ème page de couverture,
-la page d'en-tête, -la page de Copyright,
-les 7 pages du Sommaire,
-les 2 pages de la préface,
-les 8 pages de l'introduction.

Fiche rédigée le 22/6/2020 par Jean Ricodeau.

LISP

A Gentle Introduction to Symbolic Computation

David S. Touretzky

ET2

LISP
David S. Touretzky

"This is an excellent introductory text... The best that I have seen for presenting LISP to non-scientists." – Alan J. Perlis, Yale University

"This is the finest introduction to LISP ever written." – Daniel L. Weinreb, Symbolics, Inc.

"A very lively, readable introduction to LISP." – K. N. King, Georgia Institute of Technology

Here is the only book on LISP written for non-programmers. It is a crucial introductory text, using the most current and advanced LISP dialects.

LISP, the preferred programming language for artificial intelligence research and cognitive science, is easier to teach than Pascal or FORTRAN and more powerful to program in than BASIC. LISP's interactive and intuitive nature makes it an especially good language for those first learning about computers and programming.

Touretzky's *"gentle introduction"* to LISP – written in a clear, non-technical style – eases the reader with a non-mathematical background into programming. Each lesson is devoted to one main concept and concludes with a review of major ideas presented, followed by a list of the functions introduced. Among the outstanding features of this book are:

- ** an interactive approach that emphasizes reasoning about program behavior
- ** many examples and short pen and pencil exercises as well as "keyboard exercises" for learning on the computer terminal
- ** an excellent treatment of recursion
- ** appendices on LISP dialects and extensions to LISP

This is an ideal book for students taking their first programming course that should also appeal to computer users interested in artificial intelligence and to psychologists and psychology students working in cognitive science.

David Touretzky teaches LISP at Carnegie-Mellon University, Pittsburgh, Penn.

Harper & Row Hands On! Computer Books

Computer-aided jacket design by SULLIVAN STUDIOS

$18.95
ISBN 0-06-046657-X
01182284

LISP

A Gentle Introduction to Symbolic Computation

DAVID S. TOURETZKY
Computer Science Department
Carnegie-Mellon University

Sponsoring Editor: John Willig

HARPER & ROW, PUBLISHERS, New York
Cambridge, Philadephia, San Francisco,
London, Mexico City, São Paulo, Sydney

To Phil and Anne

Sponsoring Editor: John Willig
Project Editor: Mary E. Kennedy
Designer: C. Linda Dingler
Production: Marion Palen/Delia Tedoff
Compositor: TriStar Graphics
Cover design: Steve Sullivan
Art by: Kim Llewellyn

Library of Congress Cataloging in Publication Data
Touretzky, David S.
LISP: a gentle introduction to symbolic computation.
Copyright © 1984 Harper & Row, Publishers, Inc.
Includes index.
1. LISP (computer program language) I Title.
II Title: L.I.S.P.
QA76.73.L23T67 1984 001-64'24 83-12712

ISBN 0-06-046657-X

85 86 87 88 9 8 7 6 5 4

Contents

Preface xv
Acknowledgments xvii

Introduction: Getting Acquainted 1

1. Beauty and the Bits 1
2. Practical Reasons for Learning About Computers 2
3. Computers Are Everywhere 2
4. Computers as Research Tools 4
5. Computers and Formal Reasoning 4
6. Programs Are Descriptions 6
7. Artificial Intelligence: Can Machines Really Think? 7

Chapter 1: Functions and Data 9

1. Introduction 9
2. Functions on Numbers 9
3. Integer Division 10
4. Order of Inputs Is Important 11
5. Symbols 13
6. The Special Symbols T and NIL 14
7. Some Simple Predicates 15
8. The EQUAL Predicate 16
9. Putting Functions Together 18
   - 9.1. Defining SUB2 18
   - 9.2. Defining ONEMOREP 19
   - 9.3. Using Constants in Functions 21
10. The NOT Predicate 22
    - 10.1. Negating a Predicate 23
11. Cascading of Functions 25

12. Errors 26
13. Summary 27

Advanced Topics 1 28

1. Types of Objects 29
2. Closure 29
3. Inverses 31
4. Formal Definitions Should Also Be Constructive 32
5. Real-World Sense; Computer Nonsense 32

Chapter 2: Lists 35

1. Lists Are an Important Data Type 35
2. What Do Lists Look Like? 35
3. Length of Lists 37
4. NIL: The Empty List 38
5. Internal Representation of Lists 38
6. The CAR and CDR Functions 42
   6.1. CAR and CDR of Nested Lists 44
   6.2. CAR and CDR of NIL 44
7. CONS 46
   7.1. CONS and the Empty List 48
   7.2. Building Nested Lists with CONS 49
   7.3. CONS Can Build Lists from Scratch 49
8. LIST 50
9. LENGTH 54
10. Programming with Lists 54
    10.1. Extracting an Element from a List 54
    10.2. Taking Nested Lists Apart 56
    10.3. Replacing the First Element of a List 57
11. Nonlist Cons Structures 58
12. Summary 61

Advanced Topics 2 62

1. Symmetry of CONS and CAR/CDR 62
2. CDR and Closure 63
3. Unary Arithmetic with Lists 64

Chapter 3: EVAL Notation 67

1. Introduction 67
2. The EVAL Function 67
3. Evaluation Rules Define the Behavior of EVAL 68

4. EVAL Notation Can Do Anything Box Notation Can Do 70
5. Why We Need Quoting 70
6. The Problem of Misquoting 72
7. Two Ways to Make Lists 72
8. Defining Functions in EVAL Notation 74
9. Four Ways to Misdefine a Function 76
10. Explanation of Variable Binding 77
11. Summary 79

Meet the Computer 80

1. Running Lisp 81
2. Terminal Keyboard Layout 81
3. The Read-Eval-Print Loop 82
4. Tips for Computer Users 83

First Keyboard Exercise 85

Advanced Topics 3 86

1. A Note on Lambda Notation 86
2. Functions of No Arguments 87
3. Dynamic Scoping and the Rebinding of Variables 87
4. The QUOTE Special Form 91
5. EVAL and APPLY 92

Chapter 4: Conditionals 95

1. Introduction 95
2. The IF Special Form 95
3. The COND Special Form 98
4. Using T as a Condition 99
5. Two More Examples of COND 100
6. COND and Parenthesis Errors 102
7. The AND and OR Special Forms 104
8. Evaluating AND and OR 105
9. Building Complex Predicates 106
10. Why AND and OR Are Conditionals 108
11. Conditionals Are Interchangeable 109
12. Summary 112

Advanced Topics 4 113

1. Boolean Functions 113
2. Truth Tables 114
3. DeMorgan's Theorem 115

Chapter 5: Global Variables and Side Effects 119

1. Introduction 119
2. SETQ Assigns a Value to a Variable 119
3. BOUNDP and MAKUNBOUND 121
4. Programming with Global Variables 122
5. Side Effects 124
6. Summary 126

Keyboard Exercise 127

Advanced Topics 5 128

1. The SET Function 128
2. Rebinding Global Variables 130

Chapter 6: List Data Structures 131

1. Introduction 131
2. Some Useful Predicates 131
3. General Purpose List Functions 133
   - 3.1. REVERSE 133
   - 3.2. APPEND 134
   - 3.3. NCONS 135
   - 3.4. LAST 136
   - 3.5. NTHCDR and NTH 136
   - 3.6 SUBST 137
4. Lists as Sets 139
   - 4.1. UNION 139
   - 4.2. INTERSECTION 139
   - 4.3. SETDIFFERENCE 140
   - 4.4. MEMBER 140
5. Programming with Sets 142
6. Lists as Tables 145
   - 6.1. ASSOC 145
   - 6.2. SUBLIS 146
7. Programming with Tables 147
8. Summary 151

Keyboard Exercise 152

Advanced Topics 6 155

1. EQ vs. EQUAL 155
2. Shared Structure 156

3. Destructive Operations 157
   3.1. RPLACA, RPLACD, and DISPLACE 158
   3.2. NCONC 161
4. Programming with Destructive Operations 161

Chapter 7: Applicative Operators 165

1. Introduction 165
2. The APPLY-TO-ALL Operator 165
   2.1. Manipulating Tables with APPLY-TO-ALL 166
3. Lambda Expressions 168
4. The FIND-IF Operator 169
   4.1. Writing ASSOC with FIND-IF 170
5. SUBSET 172
6. EVERY 174
7. The REDUCE Operator 176
8. Summary 178

Keyboard Exercise 179

Advanced Topics 7 183

1. Identity Values 183
2. Left vs. Right Reduction 185
3. Operating on Multiple Lists 186
4. The MAP Functions 187

Chapter 8: Recursion 191

1. Introduction 191
2. Martin and the Dragon 191
3. A Lisp Version of Martin's Algorithm 193
4. Martin Visits the Dragon Again 195
5. A Lisp Version of the Factorial Function 197
6. The Dragon's Dream 198
7. A Lisp Function for Counting Slices of Bread 199
8. The Three Rules of Recursion 200
9. Martin Discovers Infinite Recursion 203
10. Infinite Recursion in Lisp 205
11. Building Lists with Recursion 207
12. Two-Part Recursions 210
13. Recursion in Art and Literature 213
14. Summary 213

Keyboard Exercise 214

Advanced Topics 8 218

1. Structural Recursion 218
2. Tail Recursion 222
3. Recursive Data Structures 225

Chapter 9: Elementary Input/Output 229

1. Introduction 229
2. Character Strings 229
3. The MSG Special Form 230
4. The READ Function 233
5. Summary 235

Keyboard Exercise 236

Advanced Topics 9 237

1. The Lisp 1.5 Output Primitives 237
2. Defining MSG in Terms of Primitives 238
3. Printing in Dot Notation 239
4. Hybrid Notation 240
5. File I/O 241

Chapter 10: Iteration 243

1. Introduction 243
2. The PROG Special Form 243
   - 2.1. The GO Special Form 244
   - 2.2. The RETURN Function 245
   - 2.3. Programming with PROG 247
3. The LET Special Form 250
4. The DO Special Form 252
5. How to Write Elegant Lisp Programs 253
6. Summary 254

Keyboard Exercise 256

Advanced Topics 10 259

1. PROG1, PROG2, and PROGN 259
2. Defining Special Forms 260
3. Defining Macros 260
4. Functions with Arbitrary Numbers of Inputs 261

Chapter 11: Property Lists 263

1. Introduction 263
2. Establishing Properties 263
3. Retrieving Properties 264
4. Modifying Properties 265
5. Programming with Property Lists 266

Keyboard Exercise 268

Advanced Topics 11 273

1. Property Lists and Function Definitions 273
2. Special Uses for Property Lists 274

Appendix A. Recommended Further Reading 275

Appendix B. Dialects of Lisp 279

1. MacLisp, Common Lisp, and Lisp Machine Lisp 280
2. Franz Lisp 280
3. UCI Lisp and TLC-LISP 281
4. Interlisp 281
5. P-LISP 282

Appendix C. Extensions to Lisp 283

1. Simplified Definitions 283
   - 1.1. Type Predicates 284
   - 1.2. Functions on Lists 285
   - 1.3. Functions on Sets 286
   - 1.4. IF and MSG 287
   - 1.5. Applicative Operators 288
2. The MacLisp Extensions 291

Appendix D. Answers to Exercises 299

Index 375

Preface

This book is about learning to program in Lisp. About twenty-five years old, Lisp is one of the oldest computer languages, yet its importance and popularity are increasing today as never before. One reason Lisp is so important is that it is the *lingua franca* of artificial intelligence research, or "AI." AI is moving from the laboratory to the everyday world, and tremendous changes are in store for our society as a result. Lisp's popularity is also growing because it offers a friendly, interactive programming environment, an invaluable aid to beginning programmers.

When I wrote the book I had three types of readers in mind, and I would like to address each in turn:

- The humanities student taking his or her first programming course. For you, let me stress the word *gentle* in the title. I assume no prior mathematical background beyond arithmetic. Even if you don't like math, this book may teach you to enjoy computer programming. I've avoided technical jargon. There are many examples. Plenty of exercises are interspersed with the text, the answers to which may be found in appendix D.

- Psychologists, linguists, and other persons interested in AI and cognitive science. As you begin your inquiry into artificial intelligence, you will see that almost all research in this field is carried out in Lisp. This book can be your doorway to a deeper understanding of the technical literature as well as a quick introduction to the central tool of AI.

- Computer hobbyists. Several high-quality mini-Lisp systems are available today for personal computers, such as the Apple and many Z-80

based machines. These systems are adequate for learning Lisp and writing simple programs, but currently their power is limited by a lack of memory. When more powerful personal computers become affordable—for example, those with 24- or 32-bit address spaces and virtual memory capability—hobbyists will be able to run the same Lisp software that is used in today's AI research labs. The prospect of putting research-quality Lisp systems into the hands of computer hobbyists is truly exciting. This book will teach you to use the Lisp systems available today, but more important it will help you prepare for the technology that is coming tomorrow.

There has never been a standard Lisp. Throughout its history the language has been evolving and gradually maturing; the end of this process is not yet in sight. Consequently there are a large number of Lisp dialects around. Lisp 1.5 is the closest thing there is to a "standard" Lisp, but it's a 1962 dialect that lacks much of the power of modern Lisp. The major dialects today are MacLisp, Interlisp, UCI Lisp, and Lisp Machine Lisp; Franz Lisp and Common Lisp are MacLisp offshoots that are developing rapidly and will soon be important in their own right. This book is based primarily on MacLisp and Common Lisp, with a few borrowings from other dialects. Appendix C gives information on how to make virtually any Lisp system compatible with the one used in the book.

Introduction: Getting Acquainted

1. BEAUTY AND THE BITS

There are literally hundreds of books on the market today about how to program a computer. What distinguishes this one from the rest, besides the fact that it uses Lisp, is its treatment of computer programming as an *aesthetic* activity. The computer is not a musical instrument, but at a slightly more abstract level one could truthfully call it the greatest keyboard instrument of the twentieth century. Just as the piano allows us to create beauty in structured patterns of sound, the computer is a tool for finding beauty in information structures. Much of the beauty of computer science is mathematical in flavor, but don't be put off by that fact. Programming is very different from ordinary pencil-and-paper mathematics.

I believe there is something in common between what programmers do and what musicians do. Therefore in writing this book I have borrowed some ideas from the format of an introductory piano book.

There are two components to piano books: theory and keyboard exercises. Music theory is a mathematical description of some of the elements of music. It contains formal descriptions of harmony, chord structure, counterpoint, and rhythm. Computer science theory deals with another set of formal elements: information structures, control structures, algorithms, and representation languages.

Exercises are what convert purely theoretical understanding into demonstrable skill. There is a long way to go from understanding the

properties of a theoretical piano (or computer) to getting a real one to do something interesting. Pianists and programmers put in lots of practice at their respective keyboards in order to achieve proficiency. The exercises that appear throughout this book will help you improve your own programming skills, until you are a computer "virtuoso."

2. PRACTICAL REASONS FOR LEARNING ABOUT COMPUTERS

Computer courses are one of the most rapidly growing areas in the modern college curriculum. Everyone needs some knowledge of computer basics because of the major role computers play in a technological society. Computer programming is also an excellent way to develop formal reasoning skills. People have traditionally learned formal reasoning by studying mathematics (especially geometry and logic), but for many these subjects are too difficult and tedious to be enjoyable. Programming, on the other hand, is fun—even addictive. Students can say things to the computer (in a computer language), and the computer talks back to them.

A computer that talks back helps students to explore new ideas, for they can see immediately whether the ideas work or not. Students can learn, and are encouraged to learn, by experimenting and discovering new things in a personal dialog with the machine. Computers can put animation into a subject that mere pencil and paper leave abstract and lifeless.

3. COMPUTERS ARE EVERYWHERE

Computers exist in a great variety of forms, from $10 microprocessors no larger than a thumbnail to $10 million dynamos that can easily fill a house. There are many ways the most versatile of all machines is put to use in our society. Here are a few:

- *Data Processing.* Perhaps the most well-known computer application, data processing is the primary purpose for which businesses buy computers. Computers are used to automate routine paperwork by doing such things as recording customer orders, sending out bills, managing inventory, printing paychecks, and keeping personnel records.

- *Embedded Applications.* Small computers are often placed inside other machines, such as video games, pocket calculators, electronic banking machines, and programmable microwave ovens. Even some cars now include a computer: it controls carburetion, assures even braking, and provides for a more informative instrument panel.

- *Communication.* Our telephone system would be impossible without computer-controlled switching equipment. Computers also run communications satellites and electronic communication networks. One such network, the ARPANET, spans half the globe and connects over a hundred universities, research centers, and government installations. A person with access to this network can send electronic mail to anyone else on the network in just seconds, instead of the days it takes for regular mail. Airline reservation systems are another example of how computers make communication speedier.

- *Text Processing.* Computers have revolutionized the process of composing and editing text. Far more versatile than an ordinary typewriter, a computerized "word processor" lets an author correct misspellings, change wording, and cut and paste sentences or even entire paragraphs simply by pushing a button. By the way: The page you're reading now was composed, edited, typeset, and printed using a network of cooperating computers.

- *Information Retrieval.* Computers are very good at searching through a large body of information quickly. Major libraries have links to computers that can electronically search a database of medical or legal abstracts. Tens of thousands of entries can be examined in a few minutes, saving researchers many hours of manual work. Similar information retrieval techniques are used by large businesses or utility companies to immediately retrieve a customer's record when he or she calls in with a question about service. The telephone company uses computers to help directory assistance operators look numbers up quickly.

- *Personal Computing.* As their prices keep dropping and their capabilities increase, computers are moving into more and more homes. People use home computers to play games ranging from the traditional chess, backgammon, and blackjack, to Space Invaders and Pac-Man. Home computers also do useful things such as budget planning and maintaining Christmas card lists. People who learn to program their own computers have an unlimited opportunity for creative play.

4. COMPUTERS AS RESEARCH TOOLS

There are several different ways in which computers can be used as research tools. These may be classified as processing data, viewing data, and producing data.

An example of *processing* data is statistical analysis, such as linear regression or analysis of variance. Such procedures are difficult and tedious to do by hand but can be done quite quickly and accurately by standard statistical computer programs such as SPSS or BMD.

When the computer is used to *view* data, it generates an organized view by maintaining lists in alphabetical order or generating indices or cross-reference listings on demand. Sophisticated systems now accept queries such as "list the names and addresses of all employees who work in plant #3 and make more than $15,000 a year"; the computer then selects from a large file of information only those records that satisfy the request. Another way in which computers help people view data is by generating plots and graphs automatically.

A computer can *produce* data by acting out some theory or model that a researcher is investigating. A physicist, for instance, might program a computer with detailed equations describing the behavior of a nuclear reactor. Then, after the initial conditions haven been established, the computer could solve the equations and say what the reactor would do. This is both faster and cheaper than actually building the reactor. If the physicist wanted to study core meltdowns or massive cooling system failures, a computer model might be the only way to safely conduct the research.

In practice, research computing usually involves a combination of all three of these activities.

5. COMPUTERS AND FORMAL REASONING

Formal reasoning is one way of dealing with the world around us. Other reasoning styles are not "formal," but they are equally important, such as the kinds that let us interpret poetry, invent a recipe, or mentally visualize a three-dimensional object. Formal reasoning is important because it is the foundation for modern-day science and mathematics,

fields with which every educated person should be familiar.

How do we learn formal reasoning? Usually by studying logic or geometry and learning to prove theorems. A beginning student of logic might be asked to prove a theorem such as

$$((p \Rightarrow q) \Rightarrow r) \Rightarrow (p \Rightarrow (q \Rightarrow r))$$

Or, in a geometry class, the task might be to "prove that the alternate interior angles formed by a line intersecting two parallel lines are equal." The *formal* part of the process is the way in which the student is allowed to prove the truth of a theorem. One can't say, "Well, those angles look about equal in the diagram, so the theorem must be true." One can't argue from aesthetics, either, such as saying that alternate interior angles *ought* to be equal because that leads to pleasant symmetries. The only way to show formally that a statement is true is to produce a step-by-step *logical* proof that it is true.

Many people discover they have trouble proving theorems and therefore become convinced that they can't do formal reasoning. This is unfortunate, because they are cutting themselves off from a large body of fascinating intellectual material. Theorem proving is not the only way to learn formal reasoning, and it isn't even the most useful way to apply it. After all, when was the last time you needed to prove a theorem outside of a classroom? We shall consider a different approach to formal reasoning: computer programming.

In computer programming, the object is not to prove theorems but rather to write programs to make the computer do something. The difference between a program and a theorem is that a program is more lively, more animated. The computer will act out the instructions in the program, and the programmer can monitor the computer's behavior to see if it is doing the right thing.

The formal part of computer programming is in the specification of the program. Programs are written in a concise logical notation called a **programming language**. Because the computer actually interprets the program and follows out its instructions, finding mistakes in programs is far easier than finding mistakes in proofs. Also, the subject matter of elementary computer programming is generally less abstract than that of elementary mathematics, which makes programming more intuitive and easier to learn.

6. PROGRAMS ARE DESCRIPTIONS

A computer program is a *description* of something. For instance, a program can be a description of a *procedure*. Suppose we needed a procedure for making toast. If our instructions were to be read by a human being, they could be very informal. We might describe the "make toast" procedure as:

1. Go to the breadbox.
2. Remove as many slices of bread as you want pieces of toast.
3. Put the bread in the toaster and press the handle down.
4. Wait for the toast to pop up.

Any intelligent person could follow this procedure, but a computer cannot because the procedure is too vague. For example, the procedure doesn't tell how to recognize a breadbox or a toaster if you've never seen one before. It doesn't say what to do if the toaster is not plugged in. It doesn't say what to do if you want four slices of toast but the toaster can only hold two. And it doesn't say what to do if you want dark toast but the toaster produces light toast. In fact, it doesn't even say what "toast" is!

The procedures we write in Lisp will deal with abstract things called numbers and symbols, not toasters and pieces of bread, but the point is the same: our descriptions must be precise and complete if the computer is to follow them properly.

Another possible answer to the question "What do programs describe?" is that programs describe machines. Suppose we construct a computer program to figure income tax. We could think of that program as a description of a hypothetical *income-tax-figuring machine*. A computer, then, is just a general-purpose machine that can act like any special-purpose machine, given an adequate description of the special-purpose machine.

Whether we view programming as procedure-writing or machine-building, our descriptions must be carefully thought out if the computer is to interpret them successfully. Programming languages, unlike ordinary human languages, are specially designed for writing precise (yet concise) descriptions.

7. ARTIFICIAL INTELLIGENCE: CAN MACHINES REALLY THINK?

Back in the fifties and sixties it was fashionable to refer to computers as "electronic brains." This makes about as much sense as calling an automobile "mechanical feet." Today's computers, even the very largest models, are childishly simple when compared with the human brain. They do not think, feel, or experience the world any more than an automobile does—they simply calculate things, like super-fast adding machines, and display the results.

This does not mean that computers are fundamentally incapable of thought. Whether they are or not is a topic of much heated debate. However, none of the computers that exist today are anywhere near large enough or complex enough for there to be serious consideration given to their sentience.

The field of artificial intelligence, a subdiscipline of computer science, is concerned with making computers do intelligent things such as play chess, understand spoken language, solve puzzles, or diagnose diseases. Today's computers can do all of these things (with varying degrees of success), but "thought," in the deep significant human sense, is beyond them.

Exercises

1. List five ways you use or come in contact with computers in your daily life.

2. Walk into your nearest computer store (any Sears or Radio Shack will do) and ask the salesperson why you should buy a home computer. Are his or her arguments convincing? Did you buy one?

3. Consider an economist who uses a computer to model a pet theory about monetary supply. She writes a computer program, feeds it some current data, and gets back a prediction of the state of her country's economy six months from now. The prediction turns out to be dead wrong. List as many factors as you can think of that could be responsible for this.

4. Some people have argued (fallaciously) that computers will never be able to think because "after all, a computer can only do what it's told to do." Is it possible to write down a set of instructions that *tell* a computer how to think? Take a position and defend it briefly.