The Little Schemer Simplified

Deepak Venkatesh

October 10, 2025

Contents

1 Foreword **4**

2 Preface **5**

3 Toys **7**
3.1 The Law of Car 7
3.2 The Law of CDR 8
3.3 The Law of CONS 8
3.4 The Law of NULL 8
3.5 The Law of EQ 8

4 Do It, Do It Again, and Again, and Again ... **8**

5 Cons the Magnificient **8**

6 Numbers Games **8**

7 * Oh My Gawd *: It's Full of Stars **8**

8 Shadows **8**

9 Friends and Relations **8**

10 Lambda the Ultimate **8**

11 ... and Again, and Again, and Again, ... **8**

12 What Is the Value of All of This? **8**

13 Intermission **8**

14 The Ten Commandments **8**

15 The Five Rules **8**

Note:

These are my personal notes created to deepen my understanding of Lisp and programming in general. 'The Little Schemer' (4th edition) by Daniel Friedman and Matthias Felleisen is a remarkable book that teaches programming concepts in a unique and playful way. It builds from first principles using only a small set of primitives, showing how powerful ideas—such as recursion, functional programming, lambda functions, and interpreters—can be expressed using just those few building blocks.
While the book uses Scheme, I prefer to work in Common Lisp and have adapted the examples accordingly. Despite its lighthearted tone, the book is far from an easy read—it demands close attention and careful thought. My advice to anyone who wants to learn lisp is to work through David Touretzky's book titled 'Common Lisp A Gentle Introduction to Symbolic Computing' (2nd ed.).
All mistakes in these notes, whether typographical or conceptual, are entirely my own. Any misinterpretations are as well. I am still new to both Lisp and programming.

Hardware and Software used for this study

- Common Lisp SBCL version 2.4.1
- emacs
- SLIME for emacs
- Macbook Pro 2019 2.6 Ghz I7 Intel chip with 16 GB RAM

1 Foreword

By Gerald Sussman of MIT, co-author of the book SICP (the wizard book).

Key Takeaway: *In order to be creative one must first gain control of the medium.*

- Core skills are the first set of things required to master any pursuit.
- Deep understanding is required to visualize beforehand the program which will be written.
- Lisp provides freedom and flexibility (this is something which will only come in due course of time, as we keep learning more about programming).
- Lisp was initially conceived as a theoretical vehicle for recursion and symbolic algebra (this is the algebra we have been taught in school $(a + b)^2 = a^2 + b^2 + 2ab$).
- In Lisp procedures are first class. Procedures are essentially a varaint of functions. A mathematical function maps a given input to an output (domain - range/co-domain) but a procedure is a process to arrive at the result via computation.
- First Class basically means that procedure itself can be passed around as arguments to other procedures. Procedures can be return values. They can also be stored in data structures. A similar corollary (though not exact) is composite function which is usually taught in pre-calculus.
- Lisp programs can manipulate representations of Lisp programs - this likely refers to macros and how in Lisp code can be treated as data.

2 Preface

Key Takeaway: *The goal of the book is to teach the reader to think recursively.*

- Programs take data, apply a process on that data, and then produce some data.
- Recursion is the act of defining an object or solving a problem in terms of itself.
- The authors believe that writing programs recursively in Lisp is essentially pattern recongnition. Well I think it's true for any programming language or any programming paradigm.
- For recursive programming and studying, this book we will need only a few Common Lisp primitives, namely:
  - `CAR`
  - `CDR`
  - `CONS`
  - `ATOM` (`ATOM?` actually which is not a primitive in Common Lisp but explanation below)
  - `EQ`, `EQL`, `EQUAL`, `EQUALP` (currently unsure if all or only a few will be used)
  - `NULL`, `ZEROP`, `NUMBERP`
  - `ADD1`, `SUB1` (these do not exist in Common Lisp, we will define them when needed)
  - `AND`, `OR`
  - `LAMBDA`
  - `COND`
  - `DEFUN`
- The definitions of the above primitives I am not outlining here and will come to it as we work through the book.
- Authors advise to read this book slowly. Very slowly and delibrately. Re-read it multiple times. Every concept should be clear before going onto the next page.

- In the preface we hit the first difference between Scheme and Common Lisp. `()` in Scheme is actually different from that in Common Lisp. Scheme considers `()` only a list and not an atom. While in Common Lisp `()` is considered both an atom and a list. Therefore, to stay with the theme of this book we will define our own predicate `ATOM?` in Common Lisp which will work like the `atom?` in Scheme. `ATOM` is defined as per the Lisp Hyperspec as well as Common Lisp The Language (2nd ed.) by Guy Steele as 'The predicate `ATOM` is trye if its argument is not a `CONS`, and otherwise is false. In SBCL `ATOM` will give `T`

  ```
  (atom '())
  » T
  ```

  We define our own predicate `ATOMP`

  ```
  (defun atom? (x)
    (not (listp x)))
  ```

  So now `ATOMP` for our studies of this book will give `NIL` when tested for an empty list `'()`

  ```
  (atom? '())
  » NIL
  ```

3 Toys

3.1 The Law of Car

Key Takeaway: *The primitive CAR is defined only for non-empty lists. The CAR is the first atom (element) of that list.*

- In Common Lisp an `ATOM` is anything which is not a `CONS`.
- `ATOM` will include single characters, strings, numbers, special characters.
- Anything enclosed in parenthesis/brackets `()` is a list.
- We can have nested lists which are also called improper lists and non-nested lists which are proper lists.
- An S-expression which stands for Symbolic Expression is any Lisp object that can be read and evaluated by the Lisp reader. S-expressions include both `ATOM` and `CONS` (which is used to make lists).
- Q. How many S-expressions are in the list `(how are you doing so far)` and what are they? The book answers 6 and those are the elements in the lists, basically the 6 atoms inside the list. But the list itself is an S-expression in Common Lisp so there are actually 7 S-expressions.
- The next 2 questions build up on this contradiction in my opinion. A question asks how many S-expressions are in the list `(((how) are) ((you) (doing so)) far)` and gives the answer as 3. It refers to the 3 lists inside the outermost list. So a list is an S-expression for this question but a list was not an S-expression for the prior question. Furthermore 3 should not be the correct answer here. The answer should be 12 in my opinion - 6 atoms (the words), 6 lists (nested and outermost).
- The difference of `()` again comes up since it is both a list and an atom in Common Lisp unlike Scheme. The `CAR` of `()` will be `NIL` in Common Lisp unlike Scheme. In Common Lisp as per the standards and empty list's `CAR` and `CDR` are both `NIL`.

```
(car ())
» NIL
```

- `CAR` is the first atom/element of a list. If we try to find the `CAR` of a string of character or numbers SBCL will give us a variable unbound error or say that the number is not of the type list.

3.2 The Law of CDR

Key Takeaway: *The primitive CDR*

3.3 The Law of CONS

3.4 The Law of NULL

3.5 The Law of EQ

4 Do It, Do It Again, and Again, and Again ...

5 Cons the Magnificient

6 Numbers Games

7 * Oh My Gawd *: It's Full of Stars

8 Shadows

9 Friends and Relations

10 Lambda the Ultimate

11 ... and Again, and Again, and Again, ...

12 What Is the Value of All of This?

13 Intermission

14 The Ten Commandments

15 The Five Rules