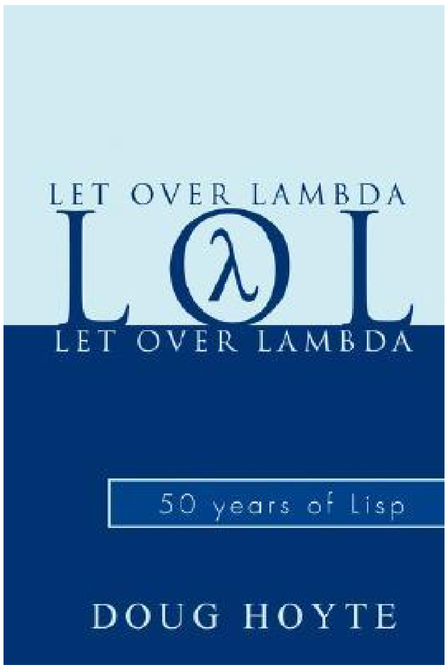

Let Over Lambda

Doug Hoyte

Let Over Lambda

Doug Hoyte

Let Over Lambda Doug Hoyte
Let Over Lambda is one of the most hardcore computer programming books out there. Starting with the fundamentals, it describes the most advanced features of the most advanced language: Common Lisp. Only the top percentile of programmers use lisp and if you can understand this book you are in the top percentile of lisp programmers. If you are looking for a dry coding manual that re-hashes common-sense techniques in whatever langue du jour, this book is not for you. This book is about pushing the boundaries of what we know about programming. While this book teaches useful skills that can help solve your programming problems today and now, it has also been designed to be entertaining and inspiring. If you have ever wondered what lisp or even programming itself is really about, this is the book you have been looking for.

Let Over Lambda Details

Date	:	Published April 2nd 2008 by Lulu.com (first published April 1st 2008)
ISBN	:	9781435712751
Author	:	Doug Hoyte
Format	:	Paperback 384 pages
Genre	:	Computer Science, Programming, Science, Technology, Computers

From Reader Review Let Over Lambda for online ebook

Michael says

This book will make your brain melt. Embrace this.

Ajit says

Definitely opinionated. Informative.

Leonardo says

Amazing book. An eye opener and a mind bender.

My highlights go to the chapters on Pandoric Macros, Compiler Macros and My favorite, Lisp Moving Forth Moving Lisp.

Read this book with an open mind though. The author is set in his ways and is highly opinionated. If you accept that, you'll have a great time reading this.

Luke says

Nerdiest book I've read in years, all about metaprogramming with macros in Lisp. Quite enjoyable, and once I got the various quotes and backquotes sorted in my mind remarkably easy reading - the example cases are well-motivated. Don't expect to write much lisp anytime soon, but a fun metaprogramming book. Most of the humor comes from making fun of Scheme. Appendix on Emacs vs Vi takes position you'd expect once you remember that elisp hasn't had lexical scope until this year...

Ronny says

I originally read this book in the hope that I would 'get' Lisp. Having tried it a few times in the past, I failed to see what was special about it. Indeed, it struck me as an awkward language with a terrible syntax. But maybe I needed a paradigm shift to really think in Lisp. This book did not provide that shift.

I didn't finish this book, and I may leave it unrated once I figure out how to clear ratings since I'm not sure I can fairly evaluate it. Frankly, I may be this book's anti-audience, and Mr. Hoyte may be my nemesis :). For instance...

I think Lisp is outdated. Yes, it was great in its day. Yes, it was better than many later languages. And yes, it contributed to many languages and showed how a simple language based on a small set of powerful concepts can outdo significantly more complex languages. But time has moved on. Better languages are available and there's no way around it, Lisp's syntax stinks. Indeed, many of Lisp's supporters seem to be unaware of (or intentionally ignore) language developments over the last decades. Hoyte does the same when he compares Lisp to C. C is outdated and stunk even when it was first released. One would have to try to make a worse language. Why not perform a detailed analysis of Lisp versus a more recent language?

I think macros are to be used sparingly -- if at all. Hoyte is a big fan of macros, but never makes a good case for them. His unit conversion example uses macros to avoid quoting the symbol. This is unconvincing for two reasons. The first is that the symbol would have been unnecessary with a different functional or object composition, and indeed would have resulted in more readable code. The second is that using macros just to avoid symbol quoting is no reason to use something as dangerous as a macro. Indeed, Hoyte even points out the risk in the double evaluation of a squaring macro, but then he goes on to resolve it with more complications rather than abandoning macros entirely.

I think Lisp is best used in a functional style. In this way, many of Lisp's more awkward features (like the set and loop forms) are avoided. This of course requires a tail recursive version of Lisp (like Scheme). Hoyte stresses that Lisp is not functional, which while true, ends up enforcing a programming style in which Lisp is even less suited.

I think Lisp is much inferior to many of the more recent languages like Haskell. For instance, Haskell outdoes Lisps Let-Over-Lambda style by giving it for free to the user (and with more flexibility) thanks to currying. Haskell outdoes Lisp in list processing by offering list comprehensions and pattern matching. In addition, Haskell's type safety is a good thing. Yet Hoyte thinks that Haskell is a language that one learns on the road to Lisp! He also doesn't like Haskell's type safety. Indeed, this seems to be a philosophical issue among us, as Hoyte has good things to say about Perl, a language which I think is only good for small, niche, throwaway projects. His love of Forth also stresses this difference, although in all fairness I think Forth is compelling because it at least offers the paradigm shift of point-free programming; but even then it has been superseded by better point-free languages like Factor.

In short, Hoyte is a Lisp fan with a classic 'hacker' mindset; he enjoys programming close to the metal and does not like language restrictions getting in his way. I think Lisp is outdated and believe strong language controls are important to control human fallibility. Many of my objections to this book are due to the differences in philosophy and therefore I wonder if it is fair that I rate this book.

Mikhail says

This book seems to be Practical Common Lisp volume 2.

Read it just after PCL, if you still not understood, why there is so much buzz about Common Lisp and macros.

Nice to found that some macros explained in Let Over Lambda are pre-built in Clojure

---

Marvin says

This was a tough one ..

Which is a pity since it contains some very interesting stuff about macro programming.

So what's in the book?
It is a book about Lisp but neither an introduction to nor a reference. It is focused on macro programming. On the way there it starts by talking about scope and closures, followed by increasingly advanced macro topics: extending the language, extending the parser, macros that build macros, interaction of multiple levels of macros and macro efficiency. This tour of macros is followed by a case study introducing a Forth implemented in Lisp using the lessons learned earlier.

And why did I find it tough?
The problem was not the wealth of information presented but the way it was presented: the author is a Common Lisp zealot and keeps on preaching its superiority. I find that hard to take.

Jonathan Avery says

This book is excellent and lived up to expectations.

That being said, the benchmark between sortf and CL's sort on page 280 is misleading. He neglected to include the time of creating the function used in the benchmark. This includes what looks to be an expensive call to build-batcher-sn.

This specialized sort has a place: when you have data-sets of fixed size, and you can amortize the time spent on building the function with the time saved on its repeated use. His macro sortf implies creation of the lambda form for a one time use, this could be a bad idea.

If anyone here does a formal algorithmic analysis of build-batcher-sn on page 262, please let me know. A crude intuition would lead me to believe it is at least O(n^k) where k is > 2.

Derek Verlee says

Very interesting tour of meta-programming with macros in lisp (among other things).
Definitely perked my interest with regards to features such as macros that do well supported many of the more popular languages, with regards to productivity or even capability.

Jake McCrary says

This book is a deep dive into a using Common Lisp macros. Doug delves into some complicated types of macros which allow for some pretty powerful ideas to be expressed simply.

The last chapter of the book (where he implements a macro which takes Forth and runs it) is mind bending.

Not a book for someone who is not familiar with macros. Worth reading for anyone interested in being exposed to complicated macros. I can only take the author's word that this is what being a Lisp professional is about.

Alex Nelson says

This is probably the best book on macros I've read, and if you're programming in any LISP...you should read this book.

That said, there are a number of problems I have with this book. For example, section 3.7 "Duality of Syntax" --- the author never specifies what he means by "duality", nor does he clearly express what this "duality of syntax" *is*. But apparently it's important, and it intentionally violates referential transparency. How is this a duality? Does the author mean that static & dynamic contexts are treated as "the same", and therefore this seemingly arbitrary distinction inherited from Blub languages *is* a "duality"?

This aside, the book really is worth reading. The only caution I would have is for a Clojure coder to pick it up, I would just suggest to learn some Common Lisp too (y'know, not just to be a well-rounded individual, but to understand what's going on in the book!).

Scott King says

This is a very advanced book on LISP macros. Definitely a must-read, but you have to read it slowly and thoughtfully to get all it has to offer. I expect it will be a book to dip into repeatedly over time.
