quora.com

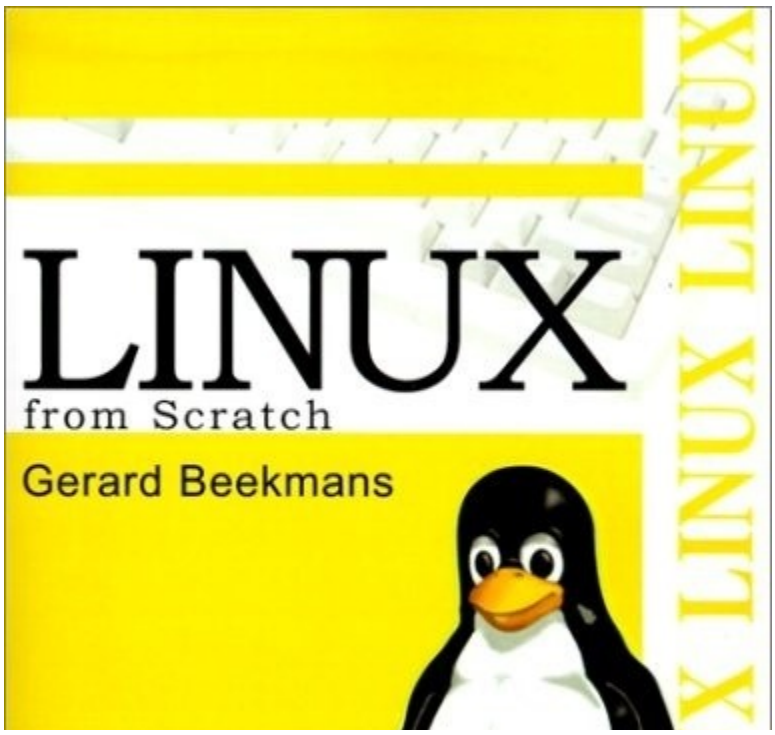
What single book has increased your programming skills the most? Why?

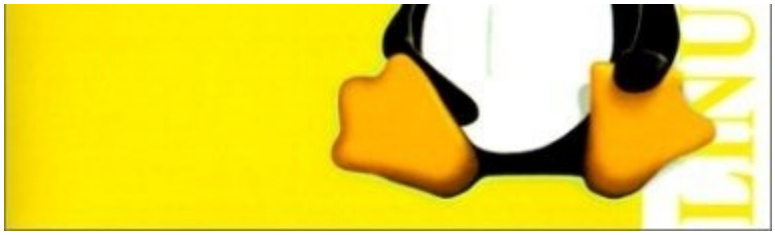
12-16 minutes

There's a book I went through in my younger years that surprisingly keeps on giving and giving. If I could pick one and only one book which I would ascribe a majority of my skills to it would be, without hesitation, this one.

Let me add to that, I'm not being hyperbolic at all when I say that anyone who goes through this book will emerge a decidedly better programmer, improving virtually *every* skill that a programmer needs in his professional life.

Without further ado:





[Linux From Scratch!]

Open-source software is one of the best sources (pun intended) of learning available to a programmer. Navigating open-source, however, can be confusing and daunting. Now it will probably take you a few months to go through this book but once you're done you'll be intimately familiar with all aspects of OSS.

What this book essentially teaches you is to build an entire Linux distro from scratch. Piece-by-piece you make your own Fedora/Ubuntu and gradually everything is demystified. Here's a partial list of reasons why this book is the bee's knees:

- **Moderate Level of Complexity**

  Sure, you can directly pick up a book about the kernel or the X window server. It will however be confusing and perplexing to the point of being of no use, especially to a beginner. LFS is not about understanding any component in detail, it won't bother you with how Qt works. What it *will* make you understand is how Qt fits in the bigger picture.

- **Improves Core Programming Skills**
  Writing code/typing on a keyboard actually consumes very little portion of a good programmer's schedule. Most of their time is spent on designing systems, understanding issues, comparing solutions or writing angry rants on Slashdot. (You think Trump is a contentious issue? Try posting a comment there in favor of

systemd.) LFS will, almost inevitably, break on you many many times. Each time you'll learn something invaluable. From debugging to linking to library loading, you'll be exposed to a shitload of issues that you will eventually face in your professional life anyway. The goal of this exercise is not to make a daily-use laptop but to understand how nuts and bolts of a Linux distro work together. These internals barely hold together properly in an official distro, your end-result will likely end up being worse than Windows 3.1 running on a Raspberry Pi. Nevertheless, it will significantly improve your fundamental programming skills, e.g., troubleshooting nasty integration issues, hacking fugly workarounds to make things work, navigating — at times endless and at times non-existent — information, getting help from OSS community and learning to curse at your machine without throwing it at the wall.

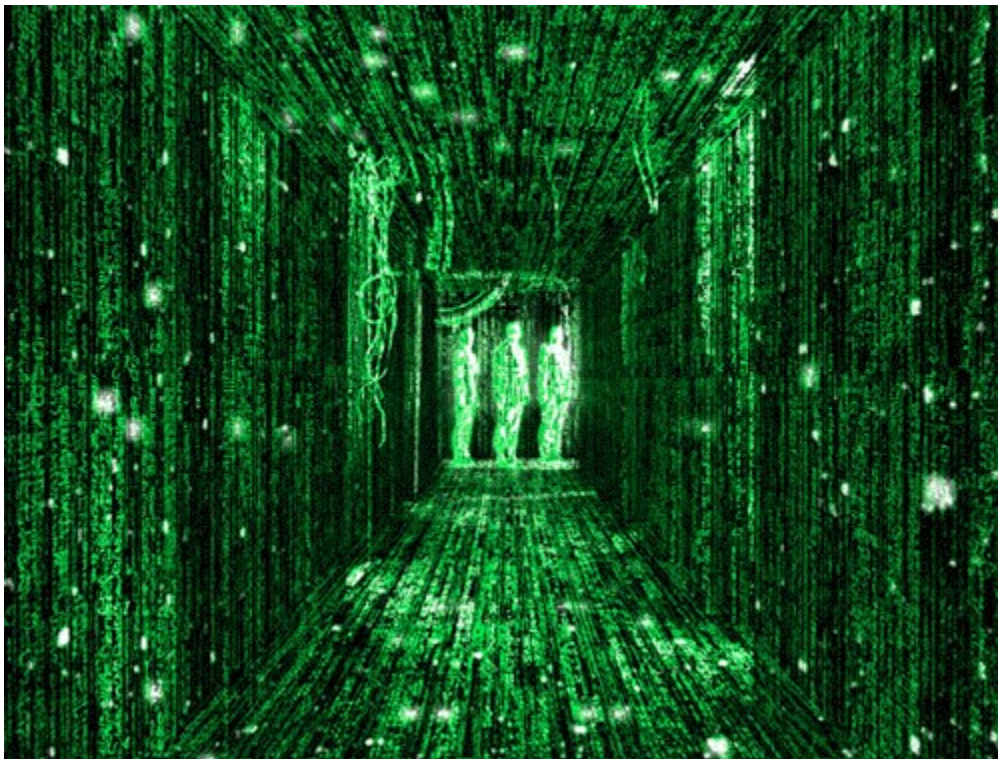
- **Teaches Long-term Skills**If you're worried about whether the things you build in LFS will be relevant by the time you're done, Don't Panic. While open-source does fragment at a scorching pace you'll be surprised at how much of the current stuff is at-its-core the same thing as before (or at least very similar to it). At some point in the future, Wayland might replace X everywhere but you'll know what a display server is for and how it is different from a widget toolkit.

  Unfortunately, it's not a trivial undertaking. You'll need lots of time and a spare machine (do ***not*** try to build LFS on a machine you care about). You also need a working knowledge of at least C, C++ and Python. If you make it through though it'll be worth it. You'll be blistering your way through codebases you thought were incomprehensible. You'll start seeing open-source as Neo sees the Matrix. You'll feel smug to the point of absurdity as someone looks

at your finished product and asks …

> “Which distro is this?”

I’ve tried to find clever replies to this one, but owing to my utter lack of creativity I usually go with “None, I built it myself.”



Gnome, KDE and Xfce: Surrounded by thousands of bugs, quietly staring at you, waiting for your next move

Id have to say the one that made the biggest impact on my programming is Clean Code by Uncle Bob(Robert C Martin)

http://www.amazon.com/Clean-Code...

Going into my first internship I had a good grasp on algorithms, problem solving, and I even though I had a good grasp on debugging....Until I had to debug other people's code. That can either be a nightmare or a blessing if the person also wrote clean code.

This book completely changed the way I approached and thought

about implementing solutions to problems because I knew other people will be looking at and reviewing my code as well. Code readability is so important.

Following that would probably be:
http://www.amazon.com/Design-Pat...

Commonly known as the Gang of Four book.

I later on purchased Head First Design Patterns and found it much more enjoyable to read and refer to.

http://www.amazon.com/Head-First...

For me, it is definitely Structure and Interpretation of Computer Programs . It is a book that is densely packed with little programming revelations.

This book teaches that programming without mutable state is pretty powerful, regardless of what your imperative background suggests you. After that, you grok how powerful functions are as an abstracting mechanism and learn to see more proper abstractions in your code. It is mind-blowing how far you can go with simple abstracting mechanism being used properly.

It teaches that there may be no need to learn dark corners of a programming language syntax to use it comfortably (and teaches to appreciate good and expressive syntax at the same time). After that, all discussions about syntactic advantages of one language over another usually seem shallow and boring.

This book teaches you to construct a whole object system from mutable state and functions. After that, you know that there is no deep magic in OOP that must be put in the language itself.

This book teaches that programming languages are not that huge

mastodons usually encountered at the programmer's workplace. The very idea of changing a "serious" industrial language is terrifying, but it should not be that way. Languages may be small, compact, wholly comprehensible, easy to modify according to your needs. Languages should not be nailed to the same inflexible semantics almost all widely known programming languages are (for example, there is a place for logic/actor/nondeterministic programming when your task fits it).

This book teaches that all its high-level concepts are not so distant from the computer hardware. Writing a compiler and a run-time system for a quite high-level programming language may be rather transparent and straightforward, there is not magic there too.

For me, I think it was Jon Bentley's well-named "Programming Pearls" and the sequel "More Programming Pearls", written in the late 80's. In each chapter (which started off as CACM columns, if memory serves), Bentley would go over some absolute gem of a widely-applicable programming technique. Even several decades after reading it, some of the things he wrote have stayed with me. Some examples:

Little languages

Sometimes the best architecture is to invent a little programming language. Bentley's example was the Pic language, used by Brian Kernighan as part of the graphics programs that came with UNIX. His clear presentation of all of the parts of implementing an ad hoc language was a true light-bulb moment for me.

Bitmap sort

Bentley started his book with a practical problem that involved sorting. His journey of exploring the characteristics of the problem

lead him to a single-pass sorting algorithm that matched the use case brilliantly, and sorted the input in a single pass.

The individual examples are wonderful, but more important, Bentley's books offered me a new way of thinking about programming, and gave me the encouragement that I needed to explore and apply it.

When I was a beginning programmer back in th 1970's, I needed to develop a program that traversed a network of structures and keep track of them and print out the result. I had learned programming doing Numerical Methods for numerical calculations in Fortran. Fairly straightforward brute force Programming. Fortran wasn't capable of handling this. So, the only other language available (it was a military project) was JOVIAL (a take off on Algol). I neede to understand recursion, data structures and how they all fit together. So, I found Nicholas Wirth's book Algorithms and Data Structures: Niklaus Wirth: 9780130220059: Amazon.com: Books. This was my first. REAL introduction to what I would call Computer Science. I loved it. The examples were written in Pseudocode, but all the major data structures and the algorithms associated with them were described. There have been many more, but for a guy with no formal CS training, this was my first and it radically changed the way I looked at programming…. it started my over 40 year career in software and computers.

BTW… The book is still on my shelf……

I have recently completed my B.Tech in Computer Engineering from Zakir Husain college of engineering and technology, Aligarh Muslim University, and believe me if you want to get your programming skills better than average than there is no books that can help. This

particular skill requires only a lots of practice. Although you can consider your curriculum books to understand the basic constructs of programming and data structures.

Your level of programming will only improve by practice. In my case, all the weekly assignments that we used to get in various programming labs throughout B.Tech, every student had two options either to copy from the internet or try to write the code himsel. From the very beginning I always tried to give my 110%. Tried to present the program in the best possible way, that other students found unnecessary. My extra effort yielded hardly 0.5 marks extra on the paper, but I knew what it was helping me with. My curiosity and level of understanding was increasing. I soon started taking interest in Android Development and sooner in Web Development. I was able to maintain an overall grasp of everything around.

In conclusion, refer books or web material just to get the knowledge of programming basics, the rest is completely up to you. How far you want to go with it.

All the best!

My Github Profile:https://github.com/msaqib4203

My all social media links: Hello!

The complete standout book for me was "The Zen of Assembly Language" by Michael Abrash (1990) http://www.jagregory.com/abrash-... . Only one other book comes even close, and that is discussed next in this message. Zen introduced the Intel instruction set as an opportunity to be surprised and to play. Abrash introduced me to the concept (cribbed from Heinlein) of "There ain't no such thing as the fastest code." Abrash shows

dozens of different ways of doing the same thing, showing the consequences of the choice. Anyone who has not read this book is likely suffering under the constraints of standard programming practices. Abrash clearly loves programming, assembly, and optimization, and fellow enthusiasts. If you want to experience passion second-hand (and if you experiment in his style, first-hand), READ THIS BOOK cover to cover, keep it in your library, and read it again from time-to-time.

The second most important book was "Imperfect C++" by Matthew Wilson (2005). This book illustrates how the C++ language has innumerable issues and dark corners. The idea that a language (C++ in particular) has purity is thoroughly debunked. The inference I draw is that one is always in a state of combat with the language one chooses, and that both defensive and offensive programming have their place in development.

Back in the 70s, two languages were compared (LISP and APL). With great reverence, LISP was called a "ball of mud" and APL was called a "crystal". Both languages have modern descendents (commonlisp/clojure and R/MatLab amongst others). Although crystalline, APL also had dark corners.

I agree with other people who answered that there are very valuable programming books out there. Amongst these, I recommend "The C++ Programming Language, Special Edition" by Bjarne StrouStrup 2013. I have dogeared 3 copies to oblivion, and am on my 4th.

As important as Stroustrup's book is to me, it documents an abstraction and doesn't give the hard-knocks, hands-on, real-world feel that the other two books give.

As a last nudge in the ribs, I give Intel kudos for constantly changing the iAPX86 chip such that no book, not even Abrash's, can possibly do justice to the mad world of optimization on their chips.

My answer would be Javascript Enlightenment. Not because of the content of the book´(which is excellent btw) but because it unlocked all of the fears and misunderstandings I had, as a Java programmer, with regards to learning Javascript. After that, I decided to learn JavaScript. So I could have a balanced opinion of the pros and cons of this language. I hated 90% of it. I do not want to work with it. But the last 10% I learnt & found interesting and new for me (event based programming, callbacks, closures) were a game changer. Today I am still a Javaist, but I use a LOT of javascript patterns in my Java code.

Next step: Haskell !!!