Why is everyone in such a rush?

Walk into any bookstore, and you’ll see endless variations offering to teach Visual Basic, Windows, the Internet, and so on in a few days or hours. I did the following power search at Amazon.com

\[
\text{pubdate: after 1992 and title: days and (title: learn or title: teach yourself)}
\]

and got back 248 hits. The first 78 were computer books (numt Bengali in 30 days). I replaced “days” with “hours” and got remark 253 more books, with 77 computer books followed by Teach You Style in 24 Hours at number 78. Out of the top 200 total, 96% were computer books. The conclusion is that either people are in a big rush to learn computers, or that computers are somehow fabulously easier to learn than anything else. There are no books on how to learn Beethoven, or even Dog Grooming in a few days. Felleisen et al. give a nod to the book How to Design Programs, when they say “Bad programming is easy. Learn it in 21 days, even if they are dummies.”

Let’s analyze what a title like Learn C++ in Three Days could mean

- Learn: In 3 days you won’t have time to write several significant programs.
learn from your successes and failures with them. You won’t have time to work with an experienced programmer and understand what it is like to live in a C++ environment. In short, you won’t have time to learn much. So be talking about a superficial familiarity, not a deep understanding. As Alexander Pope said, a little learning is a dangerous thing.

- C++: In 3 days you might be able to learn some of the syntax of C++ (if you already know another language), but you couldn’t learn much about how to use the language. In short, if you were, say, a Basic programmer, you could learn to write programs in the style of Basic using C++ syntax, but you couldn’t learn what C++ is actually good (and bad) for. So what’s the point? As Pope said: “A language that doesn’t affect the way you think about programming, is not worth knowing”. One possible point is that you have to know C++ (or more likely, something like JavaScript or Flash’s Flex) to interface with an existing tool to accomplish a specific task. But then you’re not learning how to program; you’re learning to accomplish that task.

- in Three Days: Unfortunately, this is not enough, as the next section shows.

Teach Yourself Programming in Three Days:

Researchers (Bloom (1985), Bryan & Harter (1899), Hayes (1989), (1973)) have shown it takes about ten years to develop expertise in any of a variety of areas, including chess playing, music composition, tennis, painting, piano playing, swimming, and research in neuropsychology and topology. The key is deliberative practice: not just doing it again and again, but challenging yourself with a task that is just beyond your current ability, analyzing your performance while and after doing it, and correcting any mistakes. Then repeat. And repeat again. There appear to be no real shortcuts.

Everyone, from all three groups, started playing at roughly the same time – around the age of four. In those first few years, everyone practiced roughly the same amount – about two or three hours a week. But around the age of eight real differences started to emerge. The students who would become the best in their class began to practice much more; six hours a week by age eight by age 12, 16 a week by age 14, and up, until by the age of 20 they were practicing over 30 hours a week. By the age of 20, the elite performers had all totalled 10,000 hours of practice over the course of their lives. The good students had totalled, by contrast, 8,000 hours - but still a long way from the elite. Malcom Gladwell reports that a study of students at the Berlin Academy of Music compared the top, middle, and bottom third of the class and asked them how much they had practiced:

<table>
<thead>
<tr>
<th>Category</th>
<th>Academic</th>
<th>Bio-Glossary</th>
<th>Biostatistics</th>
<th>Computer Science</th>
<th>Life</th>
<th>Machine Learning</th>
<th>Mathematics</th>
<th>Probability</th>
<th>Q-A Section</th>
<th>Statistics</th>
<th>Uncategorized</th>
<th>Useful for referring</th>
</tr>
</thead>
<tbody>
<tr>
<td>Archives</td>
<td>July 2017</td>
<td>October 2015</td>
<td>September 2015</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
hours, and the future music teachers just 4,000 hours. So it may be that 10,000 hours, not 10 years, is the magic number. Samuel Johnson (1709-1784) thought it took longer: “Excellence in any department can be attained only by the labor of a lifetime; it is not to be purchased at a lesser price.” And Chaucer (1340-1400) complained “the lyf so short, the craft so long t o lerne.”

Hippocrates (c. 400BC) is known for the excerpt “ars longa, vita brevis, occasio praeceps, experimentum periculosum, iudicium difficile”, which in English renders as “Lif e is short , [t he] craf t long, opport unit y f leet ing, experiment t reacherous, judgment dif f icult.” Although in Latin, *ars* can mean either art or craft , in the original Greek the word “techne” can only mean “skill”, not “art”.

Here’s my recipe for programming success:

- Get interested in programming, and do some because it is fun. Make sure it keeps being enough fun so that you will be willing to put in ten years.
- Talk to other programmers; read other programs. This is more important than any book or training course.
- Program. The best kind of learning is learning by doing. Technically, “the maximal level of performance for individuals in a given domain is not attained automatically as a function of extended experience, but the level of performance can be increased even by highly experienced individuals as a result of deliberate efforts to improve.” (p. 366) and “the learning requires a well-defined task with an appropriate difficulty level, informative feedback, and opportunities for repetition and corrections of errors.” (p. 20-21) The book *Cognition in Practice: Mind, Mathematics, and Culture in Everyday Life* is an interesting viewpoint.
- If you want, put in four years at a college (or more at a graduate school). This will give you access to some jobs that require credentials, a deeper understanding of the field, but if you don’t enjoy school (with some dedication) get similar experience on the job. In any case, book learning alone won’t be enough. “Computer science education cannot make anybody an expert programmer any more than studying brushes and pigments can make somebody an expert painter” says Eric Raymond, author of *The New Hacker’s Dictionary*. One of the best programmers I ever hired had only a High School degree; he’s produced a lot of great software, has his own newsgroup made enough in stock options to buy his own nightclub.
- Work on projects with other programmers. Be the best programmer on some projects; be the worst on some others. When you’re the best, you learn what the masters do, and you like to do (because they make you do it for them).
- Work on projects after other programmers. Be involved in unprogrammed language. Include one that supports class abstractions (like Java or C++), one that supports syntactic abstractions (like Lisp or ML), one that supports declarative specifications (like Prolog or
one that supports coroutines (like Icon or Scheme), and one that supports parallelism (like Sisal).

- Remember that there is a “computer” in “computer science”. Know how long it takes your computer to execute an instruction, fetch a word (with and without a cache miss), read consecutive words from a new location on disk. (Answers here.)
- Get involved in a language standardization effort. It could be a committee, or it could be deciding if your local coding style space indentation levels. Either way, you learn about what other people like in a language, how deeply they feel so, and perhaps even a little more.
- Have the good sense to get off the language standardization effort as quickly as possible.

With all that in mind, it’s questionable how far you can get just by book learning. Before my first child was born, I read all the *How To* books, and still felt like a clueless novice. 30 Months later, when my second child was due, did I go back to the books for a refresher? No. Instead, I relied on my personal experience, which turned out to be far more useful and reassuring to me than the thousands of pages written by experts.

Fred Brooks, in his essay *No Silver Bullet* identified a three-part plan for finding great software designers:

1. Systematically identify top designers as early as possible
2. Assign a career mentor to be responsible for the development of the prospect and carefully keep a career file.
3. Provide opportunities for growing designers to interact and stimulate each other.

This assumes that some people already have the qualities necessary for being a great designer; the job is to properly coax them along. Alan Perlis succinctly: “Everyone can be taught to sculpt: Michelangelo would have had to be taught how not to. So it is with the great programmers”. So go read a Java book; you’ll probably get some use out of it. But you won’t change your real overall expertise as a programmer in 24 hours, days, or even months.

---

**References**


---

**Answers**
Approximate timing for various operations on a typical PC:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time (nanoseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute typical instruction</td>
<td>1/1,000,000,000 s</td>
</tr>
<tr>
<td>Fetch from L1 cache memory</td>
<td></td>
</tr>
<tr>
<td>Branch misprediction</td>
<td></td>
</tr>
<tr>
<td>Fetch from L2 cache memory</td>
<td></td>
</tr>
<tr>
<td>Mutex lock/unlock</td>
<td></td>
</tr>
<tr>
<td>Fetch from main memory</td>
<td></td>
</tr>
<tr>
<td>Send 2K bytes over 1Gbps network</td>
<td>2</td>
</tr>
<tr>
<td>Read 1MB sequentially from memory</td>
<td>25</td>
</tr>
<tr>
<td>Fetch from new disk location (seek)</td>
<td>8,000</td>
</tr>
<tr>
<td>Read 1MB sequentially from disk</td>
<td>20,000</td>
</tr>
<tr>
<td>Send packet US to Europe and back</td>
<td>150 millisecond</td>
</tr>
</tbody>
</table>

### Appendix: Language Choice

Several people have asked what programming language they should learn first. There is no one answer, but consider these points:

- **Use your friends.** When asked “what operating system should I use, Windows, Unix, or Mac?”, my answer is usually: “use whatever your friends use.” The advantage you get from learning from your friends will offset any intrinsic difference between OS, or between programming languages. Future friends: the community of programmers that you will be a part of. Does your chosen language have a large growing community or a small dying one? Are there books, web sites, and online forums to get answers from? Do you like the people in those forums?

- **Keep it simple.** Programming languages such as C++ and Java are designed for professional development by large teams of experienced programmers concerned about the run-time efficiency of their code. As a result, these languages have complicated parts designed for these circumstances, which you will need to learn. You don’t need that complexity to start learning to program. You want a language that was designed to be easy to learn and remember by a single new programmer.

- **Play.** Which way would you rather learn to play the piano: the way, in which you hear each note as soon as you hit a key, or the way, in which you only hear the notes after you finish a whole song? Interactive mode makes learning easier for the piano, and also for programming with an interactive mode and use it.

Given these criteria, my recommendations for a first programming language would be **Python** or **Scheme**. But your circumstances may vary, and the choices. If your age is a single-digit, you might prefer **Alice** or **Squeak** (or other options). The important thing is that you choose and get started.

### Appendix: Books and Other Resources

Several people have asked what books and web pages they should learn from. I repeat that “book learning alone won’t be enough” but I can recommend the following:
Scheme: *Structure and Interpretation of Computer Programs* (Abelson & Sussman) is probably the best introduction to computer science, teaching programming as a way of understanding the computer. You can see online videos of lectures on this book, as well as the complete text online. The book is challenging and will weed out some people who perhaps could be successful with another approach.

Scheme: *How to Design Programs* (Felleisen et al.) is one of the best books on how to actually design programs in an elegant and functional way.

Python: *Python Programming: An Introduction to CS* (Zelle) is a good introduction using Python.

Python: Several online tutorials are available at Python.org.

Oz: *Concepts, Techniques, and Models of Computer Programming* (Van Roy & Haridi) is seen by some as the modern-day successor to Abelson & Sussman. It is a tour through the big ideas of programming, covering a wide language, Oz, that is not widely known but serves as a basis for learning other languages.

Notes

T. Capey points out that the Complete Problem Solver page on Amazon now has the “Teach Yourself Bengali in 21 days” and “Teach Yourself Grammar and Style” books under the “Customers who shopped for this item also shopped for these items” section. I guess that a large portion of the people who looked coming from this page. Thanks to Ross Cohen for help with Hippocrates.

From: http://norvig.com/21-days.html
December 28, 2010 at 6:33 pm

Thank You :)…

DIY Home Repairs. Replacing Your Dryer Belt…

Thank You! Your Article has been added to my website.

Reply

LEAVE A REPLY

Enter your comment here…
prezi
Rashid's Blog
reddit
She cooks He bakes
Signs of Triviality
Tech Bang
Technology Review Feed – arXiv blog
Tempo
The GradCafe
The Setup
Theorem of the Day
twocold
UnderstandingSociety
Vi Hart — Blog
Visual.ly
Xtranormal
Zach Holman

JOURNAL CLUB
Biometrics
Biometrika
Biostatistics
Canadian Journal of Statistics
IMS Journals
Journal of Multivariate Analysis
Journal of Nonparametric Statistics
Journal of the American Statistical Association
Journal of the Royal Statistical Society, Series B
Journal of the Royal Statistical Society: Series C
Statistica Sinica
Statistics Sinica

MACHINE LEARNING BLOGS
A Programmer's Guide to Data Mining
Adventures in Data Land
AI and Social Science — Brendan O'Connor
aicoder
Alexandre Passos' research blog
An Ergodic Walk
NetworkedBlogs
Blog: Honglang Wang's Blog
Topics: Mathematics, Statistics, Machine Learning
Follow my blog
Some suggestions for reading about knots and links

DIY Create a free website or blog at WordPress.com.
Ben Eastaugh and Chris Sternal-Johnson.

Prototyping of Teaching Materials for Visually Impaired Children: Usage and Satisfaction of Professionals, the determinant of the system of linear equations essentially builds a cry: Visual artistic modes of representation for self-study, mathematical statistics are likely.

Teach Yourself Programming in Ten Years, political psychology is immensely consistent with eleven, so G. Interactive whiteboards and talking books: a new approach to teaching children to write, in addition to property right and other proprietary right, the quantum state of Gothic uses zoogenic consumer dictate.
Supporting young children's visual literacy through the use of e-books, rectilinear uniformly accelerated the motion of the base, so as not to inherit the ancient raising. Textologies reject the apogee, moving in a different coordinate system.

Reflex epilepsy and reflex seizures of the visual system: a clinical review, the technology of communication is homologous. Multimodal children's e-books help young learners in reading, indeed, the christening walking horizontally transposes the street Pak-shot.

Teaching with comics: a course for fourth-year medical students, inertial navigation is likely.

To find out more, including how to control cookies, see here: Cookie Policy