Teach Yourself Programming in Ten Years

Peter Norvig

Why is everyone in such a rush?

Walk into any bookstore, and you'll see how to *Teach Yourself Java in 7 Days* alongside 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:

```
pubdate: after 1992 and title: days and
 (title: learn or title: teach yourself)
```

and got back 248 hits. The first 78 were computer books (number 79 was *Learn Bengali in 30 days*). I replaced "days" with "hours" and got remarkably similar results: 253 more books, with 77 computer books followed by *Teach Yourself Grammar and 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 about computers, or that computers are somehow fabulously easier to learn than anything else. There are no books on how to learn Beethoven, or Quantum Physics, or even Dog Grooming in a few days.

Let's analyze what a title like *Learn Pascal in Three Days* could mean:

- **Learn:** In 3 days you won't have time to write several significant programs, and 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 that environment. In short, you won't have time to learn much. So they can only be talking about a superficial familiarity, not a deep understanding. As Alexander Pope said, a little learning is a dangerous thing.

- **Pascal:** In 3 days you might be able to learn the syntax of Pascal (if you already knew a similar language), but you couldn't learn much about how to use the syntax. In short, if you were, say, a Basic programmer, you could learn to write programs in the style of Basic using Pascal syntax, but you couldn't learn what Pascal is actually good (and bad) for. So what's the point? Alan Perlis once 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 learn a tiny bit of Pascal (or more likely, something like Visual Basic or JavaScript) because you need 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 Ten Years

Researchers (Bloom (1985), Bryan & Harter (1899), Hayes (1989), Simmon & Chase (1973)) have shown it takes about ten years to develop expertise in any of a wide variety of areas, including chess playing, music composition, telegraph operation, painting, piano playing, swimming, tennis, 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, trying it, analyzing your performance while and after doing it, and correcting any mistakes. Then repeat. And repeat again. There appear to be no real shortcuts: even Mozart, who was a musical prodigy at age 4, took 13 more years before he began to produce world-class music. In another genre, the Beatles seemed to burst onto the scene with a string of #1 hits and an appearance on the Ed Sullivan show in 1964. But they had been playing small clubs in Liverpool and Hamburg since 1957, and while they had mass appeal early on, their first great critical success, *Sgt. Peppers*, was released in 1967. A study of students at the Berlin Academy of compared the top, middle, and bottom third of the class and asked them how much they had practiced:

> Everyone, from all three groups, started playing at roughly the same time - around the age of five. In those first few years, everyone practised 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 end up as the best in their class began to practise more than everyone else: six hours a week by age nine, eight by age 12, 16 a week by age 14, and up and up, until by the age of 20 they were practising well 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 merely good students had totalled, by contrast, 8,000 hours, and the future music teachers just over 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 to lerne." Hippocrates (c. 400BC) is known for the excerpt "ars longa, vita brevis", which is part of the longer quotation "Ars longa, vita brevis, occasio praeceps, experimentum periculosum, iudicium difficile", which in

English renders as "Life is short, [the] craft long, opportunity fleeting, experiment treacherous, judgment difficult." 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 that 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. To put it more 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 most effective learning requires a well-defined task with an appropriate difficulty level for the particular individual, 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 reference for this 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, and it will give you a deeper understanding of the field, but if you don't enjoy school, you can (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 pigment 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 news group, and 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 get to test your abilities to lead a project, and to inspire others with your vision. When you're the worst, you learn what the masters do, and you learn what they don't like to do (because they make you do it for them).

- Work on projects *after* other programmers. Be involved in understanding a program written by someone else. See what it takes to understand and fix it when the original programmers are not around. Think about how to design your programs to

make it easier for those who will maintain it after you.

- Learn at least a half dozen programming languages. Include one language that supports class abstractions (like Java or C++), one that supports functional abstraction (like Lisp or ML), one that supports syntactic abstraction (like Lisp), one that supports declarative specifications (like Prolog or C++ templates), 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 from memory (with and without a cache miss), read consecutive words from disk, and seek to a new location on disk. (Answers here.)

- Get involved in a language standardization effort. It could be the ANSI C++ committee, or it could be deciding if your local coding style will have 2 or 4 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 about why they feel so.

- Have the good sense to get off the language standardization effort as quickly as possible.

With all that in mind, its 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 Bullets* 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 put it more 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 ahead and buy that Java book; you'll probably get some use

out of it. But you won't change your life, or your real overall expertise as a programmer in 24 hours, days, or even months.

References

Bloom, Benjamin (ed.) *Developing Talent in Young People*, Ballantine, 1985.

Brooks, Fred, *No Silver Bullets*, IEEE Computer, vol. 20, no. 4, 1987, p. 10-19.

Bryan, W.L. & Harter, N. "Studies on the telegraphic language: The acquisition of a hierarchy of habits. *Psychology Review*, 1899, 8, 345-375

Hayes, John R., *Complete Problem Solver* Lawrence Erlbaum, 1989.

Chase, William G. & Simon, Herbert A. "Perception in Chess" *Cognitive Psychology*, 1973, 4, 55-81.

Lave, Jean, *Cognition in Practice: Mind, Mathematics, and Culture in Everyday Life*, Cambridge University Press, 1988.

Answers

Approximate timing for various operations on a typical 1GHz PC in summer 2001:

execute single instruction	1 nanosec = (1/1,000,000,000) sec
fetch word from L1 cache memory	2 nanosec
fetch word from main memory	10 nanosec
fetch word from consecutive disk location	200 nanosec
fetch word from new disk location (seek)	8,000,000 nanosec = 8 millisec

---

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. Also consider your future friends: the community of programmers that you will be a part of if you continue. 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 who are concerned about the run-time efficiency of their code. As a result, these languages have complicated parts designed for these circumstances. You're concerned with learning to program. You don't need that complication. 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 normal, interactive way, in which you hear each note as soon as you hit a key, or "batch" mode, in which you only hear the notes after you finish a whole song? Clearly, interactive mode makes learning easier for the piano, and also for programming. Insist on a language 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 there are other good choices. If your age is a single-digit, you might prefer Alice or Squeak (older learners might also enjoy these). 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, and it does teach programming as a way of understanding the computer science. 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 Intro 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 wider range than Abelson & Sussman while being perhaps easier to read and follow. It uses a 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 look at that book are coming from this page. Thanks to Ross Cohen for help with Hippocrates.
