# 十年学会编程

这篇文章已经有很多人翻译过了，我之所以再搞一次，首先是我觉得这文章真的很好，值得一搞再搞，其次，几乎所有的版本都漏掉了一些语句，我猜可能是英文版更新过，但是，中文版没有与时俱进，最后是我没有找到那种一段英文，一段翻译的版本。如果再有人在我的公众号里问我如何学习编程的话，我就可以把这个发给他了。

# Teach Yourself Programming in Ten Years

### Peter NorviG

## Why is everyone in such a rush？

## 为何每个人都急于求成?

Walk into any bookstore, and you’ll see how to Teach Yourself Java in 24 Hours alongside endless variations offering to teach C, SQL, Ruby, Algorithms, and so on in a few days or hours. The Amazon advanced search for [[title: teach, yourself, hours, since: 2000](https://www.amazon.com/gp/search/ref=sr_adv_b/?search-alias=stripbooks&unfiltered=1&field-keywords=&field-author=&field-title=teach+yourself+hours&field-isbn=&field-publisher=&node=&field-p_n_condition-type=&field-feature_browse-bin=&field-subject=&field-language=&field-dateop=After&field-datemod=&field-dateyear=2000&sort=relevanceexprank&Adv-Srch-Books-Submit.x=16&Adv-Srch-Books-Submit.y=5) and found 512 such books. Of the top ten, nine are programming books (the other is about bookkeeping). Similar results come from replacing “teach yourself” with “learn” or “hours” with “days.”

随便走进一家书店，都会看到类似《24小时学会Java》这样的书，旁边摆着的是大量诸如在几天，几小时之内学会C语言，SQL，Ruby以及算法这样的书。如果使用Amazon上的高级搜索，使用[title: teach, yourself, hours, since: 2000](https://www.amazon.com/gp/search/ref=sr_adv_b/?search-alias=stripbooks&unfiltered=1&field-keywords=&field-author=&field-title=teach+yourself+hours&field-isbn=&field-publisher=&node=&field-p_n_condition-type=&field-feature_browse-bin=&field-subject=&field-language=&field-dateop=After&field-datemod=&field-dateyear=2000&sort=relevanceexprank&Adv-Srch-Books-Submit.x=16&Adv-Srch-Books-Submit.y=5)这个关键字来搜索，将会返回512本书。最靠前的10本书中，有9本是编程的书（唯一的例外是一本记账的书）。如果把关键字”teach yourself”替换成”learn”，或者将”hours”替换成”days”，返回的结果大同小异。

The conclusion is that either people are in a big rush to learn about programming, or that programming is somehow fabulously easier to learn than anything else. Felleisen et al. give a nod to this trend in their book How to Design Programs, when they say “Bad programming is easy. Idiots can learn it in 21 days, even if they are dummies.” The Abtruse Goose comic also had [their take](http://abstrusegoose.com/249).

很明显，人们在学习编程的时候喜欢大跃进，或者认为编程比起其它事情来，是小菜一碟的事。Felleisen在他的书《How to Design Program》里说：“编烂程一点也不难。傻瓜也可以在21天内学会，即使他们是傻逼。”在搞笑漫画网站Abtruse Goose也在[这个](http://abstrusegoose.com/249)漫画中表达了同样的看法。

Let’s analyze what a title like Teach Yourself C++ in 24 Hours could mean:

我们来看看类似《24小时自学C++》这种标题党，到底说的什么鬼东西：

* Teach Yourself: In 24 hours 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 a C++ environment. In short, you won’t have time to learn much. So the book can only be talking about a superficial familiarity, not a deep understanding. As Alexander Pope said, a little learning is a dangerous thing.
* 自学: 24小时之内，你没有时间写几个有意义的程序，也就无法从成功或失败中学到经验。24小时之内，你没有时间和一个有经验的程序员合作交流，也就无从理解使用C++编程的真实场景。简单来说，时间太短，你啥也学不到。因此，有这种干货标题的速成书，只是让你走马观花的瞧一下，绝非深入的理解。18世纪的英国诗人亚历山大·蒲柏[……1]曾说过：“一知半解”是一件危险的事情。[……1]：这个哥哥还给牛顿写了最出名的墓志铭之一：“自然和自然律隐没在黑暗中；上帝说，让牛顿去吧！遂一片光明。”
* C++: In 24 hours 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? [Alan Perlis](http://www-pu.informatik.uni-tuebingen.de/users/klaeren/epigrams.html) 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 C++ (or more likely, something like JavaScript or Processing) 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.
* C++: (如果你有一定的编程基础)，在24小时之内，能学会一些C++的语法，但是，仅此而已了。如果你以前是个Basic语言的程序员，你可以写一些语法是C++，但是风格是Basic的程序，但是，24小时之内，C++语言的精华或糟粕，你都学不到。重点是什么？(首界图灵奖得主)[Alan Perlis](http://www-pu.informatik.uni-tuebingen.de/users/klaeren/epigrams.html)曾说过：“不能影响你编程思维方式的语言，是不值得学习的。”一个更加可能的场景是，在工作中，你不得不学习C++（或者JavaScript，Processing之类）的一小部分，来调用一些原有的接口，完成某个特定的任务。这种情况下，表面上说是在学习如何编程，其实只是在应付一下手上的活。
* in 24 Hours: Unfortunately, this is not enough, as the next section shows.
* 24小时内:很不幸，24小时太少了，原因下面告诉你。

## Teach Yourself Programming in Ten Years

## 十年学会编程

Researchers ([Bloom (1985)](https://www.amazon.com/exec/obidos/ASIN/034531509X/), [Bryan & Harter (1899)](http://norvig.com/21-days.html#bh), [Hayes (1989)](https://www.amazon.com/exec/obidos/ASIN/0805803092), [Simmon & Chase (1973)](http://norvig.com/21-days.html#sc)) 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.

研究表明，想在任何领域成为专家，大概需要10年的时间，这些领域包括下棋，作曲，发电报，弹钢琴，游泳，打网球，研究神经心理学和拓扑学。关键在于有针对性的训练，而不是漫无目的重复，训练的内容要刚好超过你现有的水平，练习以后要分析你的表现，改正发现的错误，然后重复，再重复。没有什么捷径可走，即使是Mozart，一个4岁就能弹琴的音乐天才，也是用了13年的时间，才真正的写出世界级的音乐作品。在流行音乐上，Beatles好像是在1964年访美期间一夜成名的。实际上，他们从1957年就已经开始在利物浦和汉堡的小酒馆里唱歌了，他们组建乐队的时间就更早了，他们第一张取得巨大成功的专辑《佩帕军士》，是在1967年发行的。

[Malcolm Gladwell](https://www.amazon.com/Outliers-Story-Success-Malcolm-Gladwell/dp/0316017922) has popularized the idea, although he concentrates on 10,000 hours, not 10 years. Henri Cartier-Bresson (1908-2004) had another metric: “Your first 10,000 photographs are your worst.” (He didn’t anticipate that with digital cameras, some people can reach that mark in a week.) True expertise may take a lifetime: Samuel Johnson (1709-1784) said “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.” Of course, no single number can be the final answer: it doesn’t seem reasonable to assume that all skills (e.g., programming, chess playing, checkers playing, and music playing) could all require exactly the same amount of time to master, nor that all people will take exactly the same amount of time. As Prof. [K. Anders Ericsson](https://www.amazon.com/K.-Anders-Ericsson/e/B000APB8AQ/ref=dp_byline_cont_book_1) puts it, “In most domains it’s remarkable how much time even the most talented individuals need in order to reach the highest levels of performance. The 10,000 hour number just gives you a sense that we’re talking years of 10 to 20 hours a week which those who some people would argue are the most innately talented individuals still need to get to the highest level.”

马尔科姆·格拉德威尔让这个理论广为人知，虽然他说的是另一个说法，10，000个小时而不是10年(这本书有中文翻译版[异类 : 不一样的成功启示录](https://book.douban.com/subject/25863621/))。法国现实主义摄影大师亨利·卡蒂埃-布列松有另外一个标准，他说你的前10,000张照片是你最差的。(当然，这个摄影大师没有意识到数码相机的出现，很多人能在一周内就可以拍10,000张照片)真正的大师要用一生来完成，英国作家，文学评论家和诗人塞缪尔·约翰逊说过：“在任何领域取得卓越的成就都要用一生的努力，不要妄想可以投机取巧。”英国的诗人乔叟也曾经说过：”吾生也有涯 而知也无涯。”同样，公元前400年的希波克拉底也表达过类似的意思，他也认为艺术没有止境，生命却有尽头。他说的这句话有个更长的版本，翻译成英语的意思是：“生命有尽头，艺术无止境，机遇转瞬即逝，练功走火入魔，决定优柔寡断。”当然，不会有一个确定的答案告诉你需要多长时间，毕竟，假设掌握所有的技能(比如编程，下棋，跳棋和音乐)需要相同的时间也不合适，不同的人也会用不同的时间。K.Anders Ericsson教授说过：“值得注意的是，在大多数领域，即使你是个天才，要达到专家级别的高水平仍然需要时间。10，000个小时只是给你一个概念，一个天才，如果每周练习10到20个小时的时间，仍然需要数年时间才能达到专家级别的水平。”

So You Want to be a Programmer

所以你想成为一个程序员

Here’s my recipe for programming success:

这是我编程成功的秘笈:

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 your ten years/10,000 hours.

要对编程感兴趣，要从中能获得快乐。保证编程足够有趣，这样才能让你花10年的时间或者10，000个小时在上面也不觉得枯燥无味。

Program. The best kind of learning is [learning by doing](http://www.engines4ed.org/hyperbook/nodes/NODE-120-pg.html). 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)](http://www2.umassd.edu/swpi/DesignInCS/expertise.html) 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.

动手编程。最好的学习方法是边学边做。常言道：“人的最高功力并不能通过做一天和尚撞一天钟这样的重复经验获得，但是，最高的功力可以通过有针对性的日复一日年复一年的训练来获得。”并且，“最有效果的学习需要有针对性的任务，这些任务的难度还要因人而异，并且有合理的反馈渠道，并能够根据这些反馈作出相应的改进。”这些观点引用自《Cognition in Practice: Mind, Mathematics, and Culture in Everyday Life》这本书里。

Talk with other programmers; read other programs. This is more important than any book or training course.

和其他的程序员交流；阅读别人的代码。这比看书或者参加培训课程要重要。

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 your own or 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](http://www.xemacs.org/)[software](https://www.mozilla.org/), has his own [news group](https://groups.google.com/groups?q=alt.fan.jwz&meta=site%3Dgroups), and made enough in stock options to buy his own [nightclub](https://en.wikipedia.org/wiki/DNA_Lounge).

如果你愿意，在大学里花上四年时间（或者再读个研）。这能让你获得一些工作机会，也能让你在这个领域有更深入的了解。但是，如果你一上学就头痛，直接去工作也能获得相同的经验。无论是什么情况，读死书肯定是不行的。计算机科学不会让你成为编程专家，就像学习画笔和颜料不可能让你成为画家一样。这是Eric　Raymond说的，他是New Hacker’s Dictionary字典的作者。我雇佣过的最好的程序员是个高中毕业的学生，他创造了伟大的软件，Mozilla，他拥有自己的新闻组，赚到了足够多的钱买了自己的酒吧。

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).

和其他的程序员一起参与一些项目。在一些项目中成为最出色的程序员；在另一些项目中当最差劲的程序员。在当最出色的程序员的时候，要测试你领导项目的能力，用你的视野来启发别人。如果你是项目中最差的那个，要学习牛人们在做什么，如果他们不喜欢做的，你要拿来做。

Work on projects after other programmers. Understand 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 them after you.

接手并推进别人的项目。理解其他人写的代码。如果别人有没有考虑到的问题，你能理解并修复这些问题。思考并设计你的软件，让这些软件容易被他人维护。

Learn at least a half dozen programming languages. Include one language that emphasizes class abstractions (like Java or C++), one that emphasizes functional abstraction (like Lisp or ML or Haskell), one that supports syntactic abstraction (like Lisp), one that supports declarative specifications (like Prolog or C++ templates), and one that emphasizes parallelism (like Clojure or Go).

学习至少6种(我不太清楚英语中的half dozen是什么意思，是虚指还是就是6种)编程语言。这6种语言中，包括一种支持抽象类的(例如Java或者C++)，一种支持函数的(Lisp, ML或者Haskell)，一种支持语义抽象的(Lisp)，一种支持声明规范的(Prolog或者C++模板)，还有一种支持并发的(Clojure或Go)。

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.](http://norvig.com/21-days.html#answers))

记住，在“计算机科学”中在“计算机”这个词语。理解执行一条指令要花多长时间，从内存中取一个字（要考虑缓存有没有命中的情况）要多久，从硬盘中连续读字节要多久，在硬盘中寻址定位要多久？

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.

参加语言的标准化工作。这可能是有关 ANSI C++ 委员会，也可能是缩进的风格是两格缩进或四格缩进。无论如何，你能从中学到其他人对语言的喜好，也能了解到他们热爱的程度，甚至也许能知道一点他们为什么他们会这样认为？

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

知道何时能从标准化语言的工作中脱身，越快越好。

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.

很难说能从书本中学到多少东西。我第一个小孩出生的时候，我读了所有的“如何做”系列的书，但是，没有什么头绪。30个月以后，当我的第二个孩子出生，我还要重温一下那些书么？当然不用了！我都已经是老司机了，我完全靠我自己的经验，事实也证明我自己的经验要胜过那些专家们写的上千页的书。

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

弗雷德·布鲁克斯在他的书《没有银弹》中给出了三条建议：

1. Systematically identify top designers as early as possible.

尽早地系统地识别出顶级设计师

1. Assign a career mentor to be responsible for the development of the prospect and carefully keep a career file.

每个人给分配一个职业规划的导师

1. Provide opportunities for growing designers to interact and stimulate each other.

让成长中的设计师有机会互相切磋武艺

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](http://www-pu.informatik.uni-tuebingen.de/users/klaeren/epigrams.html) 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”. Perlis is saying that the greats have some internal quality that transcends their training. But where does the quality come from? Is it innate? Or do they develop it through diligence? As Auguste Gusteau (the fictional chef in Ratatouille) puts it, “anyone can cook, but only the fearless can be great.” I think of it more as willingness to devote a large portion of one’s life to deliberative practice. But maybe fearless is a way to summarize that. Or, as Gusteau’s critic, Anton Ego, says: “Not everyone can become a great artist, but a great artist can come from anywhere.”

此处假设有一些人天生有成为伟大设计师的潜质，然后，就是正确的去引导他们。艾伦·佩里斯一针见血地说：”假如每个人都可以学会雕刻，那就得教米开朗基罗哪些事不要去做。对于伟大程序员，也是如此。”Perlis认为，伟大的人有一种内在的特质，这种特质往往比接受训练更重要。但是，这些特质是从哪里来的呢？与生俱来的？还是通过后天勤奋而来的？正如 动画片《料理鼠王》里的幻象大厨Auguste Gusteau说的那样：“谁都能做饭，但只有那些无所畏惧的人才能成为大厨！”我认为“把你生命中的大部分时间花在有针对性的练习上”这种精神，看作一种自愿奉献的精神！但或许“无所畏惧”才是体现这种精神的真谛。或者，就像是《料理鼠王》里那个与 Gusteau 作对的刻薄的美食评论家 Anton Ego 说的那样：“不是任何人都能成为伟大的艺术家，不过，伟大的艺术家可以来自任何地方。”

So go ahead and buy that Java/Ruby/Javascript/PHP 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 or 21 days. How about working hard to continually improve over 24 months? Well, now you’re starting to get somewhere…

所以，尽管买本 Java/Ruby/Javascript/PHP 的书吧。你可能会从中学到点儿有用的东西。但作为一个程序员，在 21 天内或 24 小时内改变你的人生，或者提高你实际的编程水平，这是不可能的。你尝试过连续 24 个月不间断的努力提高自己么？好吧，好吧，你该上路了……