Review M1932 | Teach Yourself Programming in Ten Years

## 引言

陈皓老师，网名左耳朵耗子，他在极客时间的专栏左耳听风中发布了ARTS的任务。要求每周有一篇英语技术文章的review，目的是为了学习英语，英语作为IT技术人员来说是相当重要的，因为他是查看第一手资料的重要工具。我就以这篇文章作为我第一次打卡的开始，同时也是在简书平台的第一次写文章。此文是左耳听风专栏程序员练级攻略开篇词中推荐的一篇由Peter Norvig大神撰写的“十年磨一剑”（我是这么理解的）。此文可能在互联网上有很多兄弟翻译过了，但是作为一次ARTS任务打卡，我重新用自己的理解翻译一遍也是一次学习过程。

## 正文

# Teach Yourself Programming in Ten Years

### Peter Norvig

# 十年磨一剑

>## Why is everyone in such a rush?

## 为什么大家都想着速成？

>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](http://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."

走进任何一家书店，你会看到这类书籍“如何在24小时学会Java”等一系列的不同书籍，例如“如何在几天几小时学会C，SQL，Ruby，算法等”。在亚马逊用筛选的方式查找为标题为teach，yourself，hours，从2000年开始，结果你会发现尽然有512本此类书籍。其中前10本中有9本是关于编程的书籍（还有一本是关于如何保存书籍的）。把teach yourself换做learn，把hours换做days，你会得到类似的搜索结果。

>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](http://www.ccs.neu.edu/home/matthias/HtDP2e/index.html)\*, 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).

结论就是任何一个人都在快速学习编程，同时编程不知怎么地比学其他任何事都来得简单。Felleisen在他的书“如何设计软件”（How to Design Programs）中对这种趋势也表示赞同，同时他说“坏编程很简单。笨蛋能在21天学会他，甚至他们是白痴”。Abtruse Goose的漫画中也是这么表达的。

>Let's analyze what a title like \*[Teach Yourself C++ in 24 Hours](http://www.amazon.com/Sams-Teach-Yourself-Hours-5th/dp/0672333317/ref=sr\_1\_6?s=books&ie=UTF8&qid=1412708443&sr=1-6&keywords=learn+c%2B%2B+days)\* could mean:

让我们来分析下标题为Teach Yourself C++ in 24 Hours是什么意思

>\* \*\*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.

Teach Yourself： 在24小时里你不会有时间写出几个牛逼的程序，更别提从成功与失败中学习到经验。你不会有时间和一个有经验的程序员一起工作同时理解他是怎么在C++环境中工作的。简单来说，你没有时间学习更多。所以这本书只是肤浅的谈论，并不是深层次得理解。像Alexander Pope说地，肤浅地学习是一个危险品。

>\* \*\*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++：在24小时里你可能只能学习一下C++语法（如果你已经学过其他的语言了），但是你可能都不知道怎么使用这个语言。简单来说，如果你是一个初级程序员，你可能只能学会用基础C++语法的样式来编写程序，但是你都不能学到C++对程序实际是好还是坏。所以重点是什么？Alan Perlis曾说过：“一种语言不会影响你会程序的思维逻辑，也不值得知道”。有一种可能就是你不得不学一点C++（或者像C++和一些像JavaScrip或者Processing）因为你使用现在的工具进行交互来完成一些特殊的任务。但是你不会学会用C++来编程；而你只是学会了如何来完成这个任务。

>\* \*\*in 24 Hours:\*\* Unfortunately, this is not enough, as the next section shows.

in 24 Hours：不幸的是，24小时还不够，下一节会发现更多。

>## Teach Yourself Programming in Ten Years

十年磨一剑

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

研发者发现各个领域都需要花费十年时间来达到专家级别，包括下棋，音乐制作，图像合成，绘画，演奏钢琴，游泳，羽毛球和研究神经心理学和拓扑。关键就在针对性训练：不仅仅是一遍又一遍的做，而是让自己挑战一个超过自己现有能力的任务，尝试它，在做的时候和做完之后分析下你的表现，纠正一些错误。然后重复，再重复。没有捷径：甚至莫扎特，虽然他在4岁就是一个音乐神童，但也花费了13多年才开始创作世界级的英语。另一个例子，披头士乐队似乎一夜成为一系列榜单之首和在1964年在Ed Sullivan的一场演出进入了公众视野。但是他们之前从1957开始一直在利物浦和汉堡的小俱乐部里表演，他们之前没有大型演出，他们最关键的成功作品 “Sgt. Peppers” 是在1967年发布的。

>[Malcolm Gladwell](http://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](http://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."

Malcolm Gladwell 让我总所周知这个观点，虽然他更专注于10000小时定律，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）。法国现实主义摄影大师 Henri Cartier-Bresson (1908-2004) 有另外一个标准：“你的前10000照片是你最烂的。”（他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）。真正的大师需要一生的时间：英国作家，文学评论家和诗人Samuel Johnson (1709-1784) 说：“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Chaucer (1340-1400) ：“吾生也有涯 而知也无涯。” Hippocrates (c. 400BC) ：“他也认为艺术没有止境，生命却有尽头。” 还有一个更长的引用，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，不是一个简单的数字来最终给出答案：例如编程，下棋，音乐等并不是所有这些都需要那么多时间来掌握，同时也不是所有人都需要那么多时间。Prof. [K. Anders Ericsson] 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

## 所以你想成为一名程序员

>Here's my recipe for programming success:

这里是我成功的方法：

>\* 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.

首先，对编程要有兴趣，因为做任何事都需要快乐。确定编程会给你带来足够的快乐，这样你才会去花10年或者10000小时来做这件事。

>\* \*\*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](http://www.amazon.com/exec/obidos/ASIN/0521357349)\* is an interesting reference for this viewpoint.

编程最好的学习方式边学边做。常言道：“人的最高功力并不能通过每天机械地重复经验获得，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。” “最有效的学习方法需要一个针对特定个体适当难度的制定完美的任务，得到合理的反馈，并从反馈中有机会进行改善和修正错误。” 这些有趣的观点引用自Cognition in Practice: Mind, Mathematics, and Culture in Everyday Life这本书。

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

和其他程序员交流，读其他程序。这比从书本和训练课程学习更为重要。

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

如果你花4年时间在学校这样做。你会得到一份工作，同时也会给你一些这个领域深层次得理解。但是如果你没能在学校中这样做，你得到的经验也就这些。任何情况下，只学习书本是不够的。“计算机科学教育不会使你成为专家，就像学习画笔和颜料不会使你成为画家一样” Eric Raymond， The New Hacker's Dictionary的作者。我曾经雇佣过一个最好的程序员，他只有高中学历；他创造过许多优秀软件，也有自己的组织，同时也赚了很多钱买了自己的酒吧。

>\* 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).

与其他程序员一起参与项目。在一些项目上成为最优秀的程序员；在一些项目上成为最差的。当你成为最好的，你可以测试你的能力来领导这个项目，同时用你的视角去启发别人。当你成为最差的，你要学习达人是怎么做的，你要学习别人不想做什么，你可以拿来自己做。

>\* 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.

接手别人的项目。理解别人写的程序。理解并修复之前程序并没有发现的东西。思考如何去设计你的程序使别人后期能更好的维护。

>\* 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).

学习至少半打程序语言。包括一门面向对象语言（Java 或 C++），一门函数式编程（Lisp 或者 ML 或者 Haskell），一门语法抽象（Lisp），一门声明范式（Prolog 或者 C++ templates）和一门支持并发（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))

记住计算机科学中的计算机这个词。理解一条指令需要花费多少时间，从内存获取一个词要多久，有没有缓存丢失，从磁盘读取连续词要多久，磁盘寻址要多久。答案都在这。

>\* 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.

参加语言标准化事业。他可能是ANSI C++委员会，或者决定你本地代码风格是2个空格还是4个空格。不过如何，你要知道别人喜欢语言是怎么样的，他们有多深的感受，也许只有一点点感受。

>\* 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.

很难说能从书本总学到多少。在我第一个孩子出生以前，我读过所有如何做系列书籍，但仍然感觉乱糟糟的。30个月后，我第二个孩子的到来，我需要重读一下这些书吗？不，取而代之的是我依赖于我的个人经验，这表明比起那些上千字专家写得书来，我的经验更加有用。

>Fred Brooks, in his essay \*[No Silver Bullet](http://en.wikipedia.org/wiki/No\_Silver\_Bullet)\* identified a three-part plan for finding great software designers:

Fred Brooks在他的书 No Silver Bullet 中给软件工程师三条建议：

>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.

1. 尽早地系统地识别出顶级设计师

2. 每个人给分配一个职业规划的导师，为了能保证能有好发展的职业规划和职业生涯

3. 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."

假设一个人已经具备了成为优秀设计师的很多必备条件；那只需要正确引导他。Alan Perlis一针见血的说：“每个人都可以被教会雕刻：不得不教米开朗琪罗别去做。所以对优秀程序员来说也是一样。” Perlise说优秀的人有一些内在特质，这些内在特质比训练更重要。但是这些特质从哪里来？天生的？还是后天努力而来？正如动画片《料理鼠王》里的幻象大厨Auguste Gusteau说的那样：“任何人都可以做饭，但只有无所畏惧的人才能成为优秀的厨师。”我认为把我生命中大部分时间来做针对性训练。但是只有无所畏惧才能总结他。作为Gusteau的评论者，Anton Ego说：“不是所有人都能成为伟大的艺术家，但是一个伟大的艺术家可以从任何地方来。”

>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，去买Java/Ruby/Javascript/PHP的书吧；你可能会从书中得到有用的东西。但是在24小时或21天不可能改变你的生活，也不会使你成为编程大师。超过24个月的持续努力提升怎么样？好吧，上吧，源先生。

原文：

<http://norvig.com/21-days.html>