用十年教会自己编程

作者：Peter Norvig　　译者：刘海粟

为何万事都如此仓促？

随便走进一家书店，你就能看到《7 天学会 Java》以及各种万变不离其宗的书籍，形如：在数天或是数小时内学会 Visual Basic、Windows 系统、互联网等等。我在亚马逊做了个高级搜索：

pubdate: after 1992 and title: days and
(title: learn or title: teach yourself)[1]

共得到了 248 个结果。前 78 个是计算机类图书(第 79 个是《30 天学会孟加拉语》)。我把搜索中的“days（天)”替换成“hours（小时)”，结果得到了一个相似度惊人的结果：253 本书中，前 77 本都是计算机类图书，紧随其后的第 78 本是《24 小时内教会自己语法与格调》。而在 200 名之后，有 96%是计算机类图书。

由此可见：要么是人们都在急匆匆的学习计算机，要么就是出于某种原因计算机比其他任何东西都要好学。而没有一本书是关于如何在数日之内学会贝多芬或是量子力学甚至是狗的饲养的。Felleisen 等人在《如何设计程序》中也指出了这一趋势，他们说：“糟糕的编程非常简单，蠢货都能在 21 天的时间内学会，即便他们就是根木头都可以！”

让我们来分析一下诸如《三天内学会 C++》这样的标题意味着什么：

- **学会**：3 天的时间，你完全没有时间去完成一些大型程序，也无法从其中的成功与失败中汲取知识。你也没有时间和一个有经验的程序员一起工作并熟悉 C++环境下的开发是个什么样子。总之，你没有时间去深入的学习。所以这种书只能给你一个肤浅的认识而非深入的理解。正像 Alexander Pope 说的——一知半解是件危险的事情。
- **C++**：3 天内你可能学会一些 C++的语法(前提是你学过其他编程语言)，但你学不到如何去使用这种语言。总而言之，如果你是——比方说——一个 Basic 程序员，你或许能学会以 Basic 的风格用 C++语法编程，但你无法掌握 C++眞正的优点(或缺点)。问题何在？Alan Perlis 曾说过：“如果一种语言不能影响你的编程思路，那就不值得学习。”唯一合理的解释就是你只需要学习一丁点的 C++(类似的还有 JavaScript 或是 Flash 的 Flex)以便为了某个特定目标而去连接一个现有工具的接口。但如果这样，你就不是在学习如何编程，而仅仅是在学习如何完成你的目标而已。
- **三天内**：很不幸，正像我们在下一节中要展示的那样，这远远不够。

用十年教会自己编程

研究员们(Bloom (1985), Bryan & Harter (1899), Hayes (1989), Simmon & Chase (1973))已经指出在许多领域中想达到精通都需要花费十年左右的时间，这其中包括国际象棋、音乐创作、电报操作、绘画、钢琴演奏、游泳、网球以及对神经心理学或是拓扑学的研究。关键在于用心去练习：并非仅仅是一遍又一遍的单纯重复，而是要去挑战一个刚好高于你目前水平的目标。去尝试，并在做的时候以及完成后分析自己的表现，指出所有的错误。之后重复，再重复……这没有捷径：甚至是莫扎特——4 岁时他就是个音乐神童了，但一样是用了 13 年的时间才创作出世界级的音乐。另一个例子，虽然看上去甲壳虫乐队是在 1964 年的埃德·沙利文秀上一夜走红的，但其实他们早在 1957 年就开始在利物浦和汉堡的小俱乐

部中演出了，而且虽然很早就受到大众的青睐，但他们第一次重要的成功却是1967年发行的专辑——《Sgt. Peppers》。Malcolm Gladwell公布了一份关于柏林音乐学院学生的研究报告，比较了优等、中等和后进的三类学生并为他们制定了练习时间：

> 三组人从几乎相同的年龄开始练习演奏——大约五岁左右。起初的几年中，每个人都练习几乎相同的时间——每周2到3个小时。但到了八岁左右的时候，真正的差距开始出现了。那些班中最优秀的学生的练习量开始超过其他人：九岁时达到每周6小时，十二岁时每周8小时，十四岁时每周16小时，越来越多。到了二十岁的时候，他们每周的练习时间超过了30小时。二十岁时，在精英演奏者的人生历程中共计练习了10,000小时。相较之下，仅仅是好学生的练习时间只有8,000小时，而未来的音乐教师仅有4,000小时的练习。

所以，或许10,000小时——而不是10年——才是一个神奇的数字。Samuel Johnson (1709-1784)认为这可能要花更长的时间：“要实现任何领域的卓越才能都是需要毕生奋斗的——不会再有更低的代价来获得它了。”而Chaucer (1340-1400)则抱怨“人生短暂，但却有太多技艺要学。”Hippocrates (约公元前400年)因那句“ars longa, vita brevis”而闻名，全文是“Ars longa, vita brevis, occasio praeceps, experimentum periculosum, iudicium difficile”，这话用中文[2]表达出来既是“技艺永恒，生命短暂，机会易逝，实验诡诈，抉择艰难”。虽然在拉丁文中“ars”一词既可以表示“艺术”也可以表示“技术”，但在原本的希腊文中“techne”一词则只有“技能”的意思而非“艺术”。

以下是我编程成功的诀窍：

- 对编程产生兴趣，并试着从兴趣出发去做些什么。你要确信它能持续的给你带来乐趣好让你能够为它倾注十年的心血。
- 与其他程序员交流，阅读其他的程序。这比任何一本书或一项训练都要重要。
- 编程。学习的最好方式就是边做边学。更学术性的说：“在特定领域内，个人能力的最高上限无法通过长期经验而自动获得。但即便是经验丰富的个人也可以通过刻意的努力而获得经验的提高。”(p. 336)并且“最为有效的学习需要针对特定个体、信息反馈以及重复和改正错误的机会有一个适当难度的明确目标” (p. 20-21)。《实践中认知：日常生活中的思想，数学与文化》是一本对于该观点有趣的参考书籍。
- 如果你愿意，在大学中投入四年的时间(或者继续在研究生学院投入更多的时间)。这将使你获得一些工作的入门资质，并且会给你一些关于这个学科更加深入的认识，而如果你不喜欢上学，你也可以(需要一些贡献)在工作中获得类似的经验。但无论如何，仅仅看书是绝对不够的。“计算机科学的教育不会让任何人成为专业程序员，正如研究笔刷和颜料不会让任何人成为专业画家一样。”，《新黑客辞典》的作者Eric Raymand如是说。我所聘用过的最好的程序员之一[3]仅拥有高中学历。他却创造出了很多伟大的软件，拥有他自己的新闻组，甚至在股票期权中赚到足够的钱买下一家自己的夜总会。
- 与其他程序员共同完成一些项目。在某些项目中成为最出色的程序员，而在其他一些项目中成为最糟糕的。当你最出色时，你将有机会测试自己领导一个项目的能力，并且以你的视野去激励其他人。当你最糟糕时，你要学习大师们做了什么，而不喜欢做什么(因为他们让你去为他们做)。

- 从其他程序员那里接手一些项目。理解其他人编写好的程序。看看有什么需要理解的，并在原作者不在的时候试着自己去解决一些问题。考虑一下如何设计你的程序能让它更容易被那些从你手里接手项目的人们理解。
- 学习至少六种编程语言。其中包括一种支持类抽象的语言(如 Java 或 C++)，一种支持函数抽象的语言(如 LISP 或 ML)，一种支持语法抽象的语言(如 LISP)，一种支持声明规范的语言(如 Prolog 或 C++模板)，一种支持协程的语言(如 Icon 或 Scheme)以及一种支持并行处理的语言(如 Sisal)。
- 牢记在“计算机科学”中有一个“计算机”。你要知道计算机需要多长时间去执行你的一条指令、需要多长时间从内存中读取一个字(带有或不带有缓存缺失)、需要多长时间从磁盘中连续读取字符以及需要多长时间完成磁盘的重新定位。(答案在这里)
- 参与一个语言的标准制定工作。这可以是 ANSI C++委员会项目，也可以仅仅是决定你的代码是用 2 个还是 4 个空格作为缩进。但无论如何，这将会让你学到其他人对语言的偏好以及他们到底对此有多么偏好，甚至你还可能明白为什么他们会有此偏好。
- 拥有尽快从语言标准化工作中抽身的理智。

出于以上经验，我很怀疑你能从书本中学到多少。在我第一个孩子出生前，我读了所有的指南书籍，但依然感到茫然无措。30 个月之后，当我第二个孩子出生的时候，我还需要回去复习那些书籍么？不，这次完全凭借我的个人经验了。这对我来说显然比专家们写的数千页的纸张更有效果。

Fred Brooks，在他的文章《没有银弹》中指出了寻找一位伟大的程序设计者的三个步骤：

1.尽早系统的确定一批顶级设计者队伍。

2.指派一个业务主管来负责前景的发展以及确保职业规划。

3.为增进设计师们的互相影响与激励提供足够的机会。

这样做的前提是已经假定了某人具有一个伟大设计者所应有的素质，他要做的只是去引导其他人的前进。Alan Perlis 将这些变得更加简洁：“每个人都能学会雕塑：米开朗基罗这样的人反倒需要学习如何不去雕塑。伟大的程序员也是如此。”

所以，尽管去买那本 Java 的教程吧。或许你能从中学到些什么，但你不会因此改变人生，也不会在 24 小时、24 天甚至是 24 个月之后成为一个真正的程序员。

参考文献：

Bloom，Benjamin（编）《在年轻人中培养人才》，百龄坛，1985.

Brooks，Fred，《没有银弹》，IEEE 计算机，vol. 20，no. 4，1987，p. 10-19.

Bryan，W.L.和 Harter，N.《电报语言研究：一种习惯上的收获》，心理学回顾，1899，8，345-375

Hayes，John R.，《完全问题求解》，Lawrence Erlbaum，1989.

Chase，William G.和 Simon，Herbert A.《国际象棋的感知》，认知心理学，1973，4，55-81

Lave，Jean，《实践中认知：日常生活中的思想，数学与文化》，剑桥大学出版社，1988.

答案：

在典型 PC 机上各种操作的近似时间：

执行典型指令	1/1,000,000,000 秒=1 纳秒
从一级缓存中读取数据	0.5 纳秒
分支预测错误	5 纳秒
从二级缓存中读取数据	7 纳秒
互斥锁定/解锁	25 纳秒
从主存储器中读取数据	100 纳秒
在 1Gbps 的网络中发送 2KB 数据	20,000 纳秒
从内存中读取 1MB 数据	250,000 纳秒
从新的磁盘位置读取数据(寻轨)	8,000,000 纳秒
从磁盘中读取 1MB 数据	20,000,000 纳秒
在美国向欧洲发包并返回	150 毫秒=150,000,000 纳秒

附录：语言的选择

许多人都问我应该首先学习哪种编程语言。答案并不唯一，但需要考虑以下几点：

- 善用你的朋友们。每每被问及“我该用哪种操作系统，Windows、Unix 还是 Mac？”的时候，我的回答总是：“用你的朋友们都在用的那种。”你在朋友那里学到的东西要远大于操作系统或编程语言之间的固有优势。但也要考虑到你未来的朋友：若你继续学习，你肯定会加入到某个程序员社区中去。你所选择的语言是否拥有一个大规模发展的社区？还是濒临灭绝？是否有足够的书籍、网站或在线论坛让你寻找答案？你是否喜欢那些社区中的人们？
- 保持简单。像 C++或 Java 这种编程语言是为那些由关心他们代码执行效率且经验老道的程序员组成的大规模团队来进行专业开发而设计的。因此，这些编程语言都拥有为

这些情况而设计的复杂结构。你关心的是学习编程，而不是那些复杂的东西。所以你需要的是一种为了程序员新手便于记忆和学习而设计的语言。

- 运行。你更喜欢哪种学习钢琴演奏的方式：正常的交互模式，当你按下一个琴键的时候立刻就可以听到对应的音符。还是“批发”模式，只有在你完成整段乐曲之后才能听到声音？显然互动模式让学习钢琴变得更简单，编程亦是如此。坚持一种交互模式的语言并使用它。

给出以上这些准则，我建议首先学习的语言是Python或Scheme。但你自身的情况可能有所不同，所以还有很多其他不错的选择。如果你的年龄是个位数，你可能更喜欢Alice或Squeak（年长一些的学习者可能也会喜欢这些）。重点在于——你选择并且你开始。

附录：书籍与其他资源

常有人问有哪些书籍或网站可以学习。我重申“仅仅看书是绝对不够的”，但我可以推荐以下这些：

- **Scheme**：《计算机程序的结构与诠释》(Abelson和Sussman)可能是对计算机科学最好的介绍，同时作为了解计算机科学的一种途径，本书也讲解如何编程。你可以看这本书的在线视频讲座，或是完整的在线文本。这本书具有挑战性并将淘汰一些用其他方法取得成功的人。
- **Scheme**：《如何设计程序》(Felleisen等编)是一本非常优秀的介绍如何用典雅又不失实用的方式去编程的书籍。
- **Python**：《Python编程：计算机科学导论》(Zelle)是一部用Python完成的不错的导论。
- **Python**：Python网站上的一些在线教程都是非常实用的。
- **Oz**：《计算机编程的概念，技术和模型》(Van Roy和Haridi)被认为是当代Abelson与Sussman的继承者。本书通过编程的整体构思，在更易于阅读和学习的同时，较之Abelson与Sussman获得了更广泛的视野。该书使用了一种叫做Oz的编程语言，虽然它并不被大众所知，但可以很好的作为其他语言的基础来学习。

作者注释：

T. Capey指出，亚马逊网站上《完全问题求解》一书的页面中，“购买此商品的顾客也同时购买”一项里已经出现了《30天学会孟加拉语》和《24小时内教会自己语法与格调》。我猜这其中大部分的人是从此文中看到那些书的。感谢Ross Cohen在Hippocrates问题上对我的帮助。

译者注释：

[1]意为：出版日期在1992年之后，题目中含有“天”并同时含有“学习”或“自学”。 返回>>

[2]原文中为“in English”，是把之前的拉丁文翻译为英文。因为本文是翻译稿，所以直接说成中文。 返回>>

[3]指Jamie Zawinski——XEmacs和Netscape Navigator的创始人。 返回>>