1.     写一份自己的简历。把自己所有的相关技能都罗列出来，然后把那些在100年后还用得到的标出来。给每个技能打分，满分为10分。

2.     罗列出你所景仰的程序员。尽量包括那些与你一起工作的人，因为你会在工作中从他们身上获取一些技能。记录下他们身上的1 ~ 2个闪光点，也就是你希望自己有所提高的方面。

3.     去查看维基百科（Wikipedia.Org）上“计算机科学”栏目，找到“计算机科学的卓越先驱”这部分，从这个列表中挑选一个人，阅读他的事迹，并且在阅读时打开任何你感兴趣的链接。

4.     花20分钟通读别人的代码。读出色的代码和读糟糕的代码都是有益的，两者都要读，轮流切换。如果你无法感觉出它们之间的区别，可以求助于一位你尊敬的程序员，让他给你展示一下什么是出色的代码、什么是糟糕的代码。把你读过的代码给别人也看看，问问他们的看法。

5.     罗列出你最喜欢的10个编程工具——那些你觉得你用得最多、非有不行的工具。随机挑选其中的一个工具，花一个小时去阅读它的文档。在这一个小时里，努力去学习这个工具的某个你不曾意识到的新功能，或者发现某种新的使用方法。

6.     想一想，除了编程之外你最擅长什么事情？再想一想，你是通过怎样的锻炼才变得如此熟练和专业的？这对于你的编程工作又有什么启发呢？（怎么把这些经验应用到编程方面？）

7.     拿出一叠简历，并和一组面试官在同一个房间里待上一个小时。确保每份简历都至少被3个面试官看过，并且要给出1 ~ 3分的评分。针对那些不同面试官评判大相径庭的简历展开讨论。

8.     参与一个电话面试。事后写下你的反馈，抛出你的观点，然后与主持电话面试的人聊一聊，看看你们是否达成了一致的结论。

9.     进行一次技术面试，并且被面试的人应该是某个你不太了解的领域里的专家。让他假定听众在该领域里一无所知，因此请他从最基础的讲起。努力去理解他所说的，必要时问一些问题。

10.   有机会参与别人的技术面试。期间，你只是认真地听、认真地学。在应聘者努力解决技术问题的同时，你也要在自己脑子里尝试解决这些问题。

11.   找到一个能和你交换实际问题的人，每隔一周，相互交流编程问题。花10 ~ 15分钟来尝试解决这些问题，再用10 ~ 15分钟进行讨论（无论能否解决）。

12.   当你听到任何你一时之间也无法解决的面试问题时，赶紧回到你的座位上，把这个问题用电子邮件发给自己，以留作日后的提醒。在那一周里找出点时间，用自己最喜欢的编程语言来解决它。

我之所以喜欢Steve开出的这个清单，是因为它看上去很全面。有些程序员一想到“锻炼”，总认为就是一些编码上的难题。但在我看来，**编程更在于人，而不是代码**。因此，通过解决世上所有的、并且晦涩的编程面试题目，在提高你的个人能力方面，这种方法是有局限的。

关于“努力的学习”，我也很喜欢Peter Norvig在“Teach Yourself Programming in TenYears”（花10年时间自学编程）一文中提出的诸多建议：

1． 与别的程序员交流。读别人的代码。这比任何书籍或培训课程都更重要。

2． 动手写程序！最好的学习方法就是边做边学。

3． 在本科或研究生的课程中学习编程课程。

4． 找一些项目来做，并且需要与其他程序员形成团队来合作。在项目的进行过程中，学会辨别最出色的程序员以及最糟糕的程序员。

5． 在项目中跟随别的程序员一起工作，了解如何维护那些不是你写的代码，并且学习如何写出利于他人维护的代码。

6． 学习多种不同的编程语言，特别是那些与你现在所熟悉的语言有着不同的世界观和编程模型的。

7． 了解硬件对软件的影响。知道你的电脑执行一条指令需要多少时间，从内存中取出一个字（在有缓存或没缓存的情况下）需要多少时间，在以太网（或者因特网）上传输数据需要多少时间，从磁盘中读取连续的数据或者在磁盘上跳转到另一个位置需要多少时间，等等。