学习者指南

最近更新：07/10/2022

各位学习者，你们好！为了能够顺利有效地完成本次学习，请认真阅读这份学习指南。理论上不认真阅读本指南并不会为你带来什么损失，还有可能让你发现自己独特的学习路径，但如果仔细阅读并按照其中的提示去做，大概率会让你在未来的学习中事半功倍。

# 学习目标

作为一个为初学者（多半是零基础学习者）准备的课程，在课程结束时，一个合理的目标是你能用一两门流行的编程语言写一些实用的小程序，并以此技能作为打开软件行业大门的钥匙。

如果把一位合格的初级软件工程师所具备的知识量看做100%的话，那么这套课程的知识含量（包括硬知识和软技能的知识）大约为3%左右。学习的时候难免有疏漏，最终你到手的知识约为1%。所以，认为上完这套课程就能转行成功、找到工作是不切实际的。要知道，计算机科学专业的同学们寒窗苦读若干年都不见得一毕业就有份工作，更何况我们这样一周两次、散步式的学习呢。但如果你的终极目标是成功转行，那这套课程大概率是可以帮你迈好第一步的。

另一个有用的目标是，在完成学习后，你可以根据自己的实际情况来决策未来是不是要继续学习写代码、学多深、走多远，是不是真的要走转码这条路，还是说把写代码作为辅助当前工作的一项技能。以及判断要不要去参加培训班或报考大学的计算机科学专业。或者，你决定未来完全靠自学，那么你在这套课程中学习到的学习方法将对你产生巨大的帮助。同时我也希望大家对市面上五花八门的不实宣传具备一定的鉴别力和免疫力——不会轻易成为韭菜。

# 关于编程语言

很多同学问我：“为什么要讲xx语言、不讲yy语言？”那么，我给大家的第一条建议就是——作为一个程序员，不要跟语言绑定。你的学习重点应当是：通过对一两门语言的学习，掌握学习其它编程语言的套路。若把时间退回十几年前，一位程序员或许还可以只靠一门编程语言就把工作搞定。但现在的现实是，随便哪份工作都至少要用到三四门语言——有些语言甚至是公司内部的、连名字都没有。“项目需要用什么语言那就用什么语言，不会就拿起来学”，这才是软件工程师的常态，从初学开始就要做好心理准备。**大厂hire你，一方面是看你已经会了什么，更重要的是看你有多大的潜力去学、去做那些你还不会做的事情**。你今天所学的语言，不见得就是明天要用的语言；但你今天透过语言所掌握的各种思维方式、学习方法，则是明天一定会用、伴随你整个职涯发展的思维和方法。

# 关于初学

至今我都还记得1992年的那个夏天，我好朋友的爸爸教我们学习DOS（微软在Windows之前的操作系统），30年后的2022年，我坐在Google的办公室里写程序——这三十年里发生了什么？基本上可以概括成两种事情——有用的和没用的。我知道你们想听都发生了哪些没用的事情，这些我们留到线下聊天的时候再八卦。有用的事情里，在这三十年里贯穿始终的有三件：**兴趣**，**学习方法**，**思考方式**。不知道大家发现没有，这三样东西其实都跟技术无关，是可以迁移到任何一个行业中去的。所以我给大家的建议是：始终保持对这个世界的兴趣和好奇心，了解人类的认知是怎么回事，掌握并实践Critical Thinking。

# 关于入行

我会带你把软件开发的前、中、后端都看一遍，然后你可以选择一个切入点入行。但我的建议是大家选择偏后端作为未来的职业发展——别学我。我在前端耗了十年，最后还是投入了后端的怀抱。道理很简单：如果把一个软件工程看做一个向各个方向均衡发展的球体的话，球体的体积值远比球体的面积值大——前端相当于面积、是与其它系统交互的界面；中后端是体积，藏在里面年不见，但它是核心事务，要做的事情非常多。另外，真正让你成为一家企业的high-level工程师的，是你对业务逻辑的了解。编程语言与业务逻辑比起来要简单得多、好掌握得多。所以，不要把编程语言想象成多难学。

# 学会学习

只有学会了如何学习这个行业的知识才能在这个行业里生存下去。怎么才算在这个行业里“会学习”呢？其核心奥义就是“**满怀好奇心且独立自主**”。

“独立自主”指的是当别人用一句话告诉你一本书的名字、一个文档的链接，你可能就要花上几个小时或者几天去阅读厚达数百页的书籍或文档、在网上搜索、对代码进行debug……而且时刻意识到哪些东西是有用的、哪些东西是片面的、没用的甚至错误的。“充满好奇”指的是不放过自己周围的信息、不要轻易认为周围的信息与自己无关，听到或遇到自己不知道的东西，要发自内心地多问一个为什么，而不是被别人督促着学习。

我们的学习，会还原很多“行业的本来面目”——**你的确是初学者，但初学者不必是弱者**。我会把你当作自律的成年人和学习能力方面的强者来敬重和对待。

不要对我产生依赖。这并不是说不可以问问题，而是不要问那种课上讲的明明白白、书里写得清清楚楚、Google一下满网都是答案的问题（比如代码的错误提示）。假想你问我一个问题，我马上在视频里或者书籍里给你找到一段引用，或者马上给你Google出一堆结果，那样会很尴尬。我也会有一种被abused的感觉。但遇到好问题我会回答。比如：有启发性的问题和让大多数同学都能受益的问题，我不但会仔细回答，而且会将集中答疑的视频分享给大家。

如果有额外想学的内容，网上有数不清的书籍和视频资源——认准优秀的出版社、作者和信息源，找来学就就是了。

不要怕犯错——研究错误的根源、解决产生的问题、防止再次犯错，这些都是重要的学习方法。况且，初学者也犯不了什么惊天动地的大错。人人都会犯错，所以我们每个人都希望从错误中学习、希望别人能宽容自己的错误。那么，我们是否能够互相宽容、一起构建一个宽松且有序的学习环境呢？

学会学习还包括掌握Critical Thinking的能力，并基于这个能力提出好问题、解决各种ambiguity；精确且富有同理心地进行高效沟通……总之，这个筐很大。

# 关于学习成绩

这个课程不设任何考试，也没有任何作业。最好的成绩就是你从头到尾坚持下来、最终达到学习目标。为了取得这个成绩，要投入相当数量的学习时间。相信这个课程会让你体验到这个行业对学习扎实程度的要求。你会发现：很多时候你以为你学懂了，但实际上跟完全没懂相差无几。什么时候才会发现自己没懂？就是当知识需要拿出来用的时候。比如，当需要解决问题的时候，或者需要基于之前学过的内容推导后续内容的时候。以往学习者的经验是：课后至少复习三遍才能避免疏漏。

如果跟不上了，怎么办？没关系，课程的视频就在那里，你随时可以回来学。包括我自己，很多知识也都不是一次就学会的，而是反反复复学了好几遍才弄明白的，我又怎么能要求所有学习者必须一次学会呢？

# 关于自洽性

任何成体系的知识都是自洽的，否则逻辑推理一定会失效。这说明，无论是学习庞大的知识系统也好，还是读万卷书也好，从哪里切入都可以，并在切入后逐渐找到更好的学习方式的学习路径、把所有该学的知识都联系起来。很多时候，优良的学习方式和学习路径已经由前人总结出来了，我们只需根据自己的实际情况进行采纳即可。因为知识量的存在，所以学习所能达到的最快速度便是“不浪费时间”（或者“尽可能少地浪费时间”）。不存在能够让你绕开必学知识就能构建起自洽知识体系的“捷径”。不投入足够量的时间就获得成功，这在软件行业行不通。

# 注意细节

编程是一项非常注重细节的技能。有多注重细节？此处可省略上万字，学习过程中你肯定会体验到。我想说的是：对于一些习惯上不怎么注重细节的学习者来说，编程的一部分“难度”就来源于此。所以，希望学习编程能让你变得更加细致耐心。

# 远离傲慢、拥抱谦逊

只要你学编程，那么你迟早会遇到这样一类人：他们喜欢用一些你不知道、也用不上的知识来提问你，唯一的目的就是让你感觉自己很菜、他们水平很高。何为傲慢？傲慢就是以自我为中心、只想证明自己是对的、是高明的行为。何为谦逊？谦逊就是“做对的事情，并把事情做对”。所以，对于傲慢的人，最好的办法就是在不冒犯他们的前提下敬而远之；同时，把主要精力放在把事情做对、做好上。

谦逊的另一种表现是“包容性文化”（inclusive culture）,特别是在多样性（diversity）较强的组织里。既要“多样”，又要“包容”，这本身就是一件很有挑战的事情，所以需要我们每个人的共同努力。比如，把自己记和学习笔记分享给大家、允许大家参考并提出见解，这是包容性的体现，是值得鼓励和赞扬的；而在课堂上讨论只与个人相关的话题，就不太合适，可以线下讨论、避免浪费大家的时间。

# 免责声明

如果这套课程让你有所收获，请把它归功于你自己的努力和Google这家伟大的公司。如果我哪里讲的不好、不对，或者哪些让你感觉不满意，请将它归咎于我个人。与本课程相关的一切文本、音频、视频内容，版权和解释权匀归我个人所有。

另外，我是在一份任务很满的全职工作之余带领大家学习，每周只能拿出不多的几个小时来辅导大家。我会尽力回答每位学习者的问题，但大概率无法面面俱到。如果你的提问没有及时回复，请见谅，并再次提醒我。有共性的问题我会以文档、视频或者答疑课的形式一并回答。

# 人人都能学编程

人人都能学编程，就像人人都能弹钢琴或者绘画。但并不是每个人都有机会、有能力、有需求去成为肖邦、莫扎特，或者成为梵高、达芬奇。按你的需要去学习编程即可，没必要去跟别人比。我们这就开始。