知乎周刊-编程小白学 Python

本部分来自知乎周刊 《编程小白学 Python》的前言部分。我个人认为这是非常棒的编程入门指导。为了方便阅读，特做了截屏，方便浏览。（可能已经侵权，勿传）

转自： https://www.zhihu.com/publications/weekly/19550511

什么才算是真正的编程能力？

家飞猫

编程能力是一种解决问题的能力。如果问题没能被很好地解决，知道再多也没用。

编程能力是一种运用机器解决问题的能力。首先是要判断问题在什么程度上可被机器解决，比如理论计算机科学会告诉我们什么可做、什么理论上不可做、什么理论上可做实践上不可做。然后是让机器更好地理解问题，比如计算机都是（图灵 - 冯诺依曼模型）等价，但不同的问题可能会适用不同的编程语言。再后是让机器能更高效率地解决问题，比如同样的问题可能会有效率差别巨大的算法。

编程能力是一种抽象问题的能力。借用轮子是很好的办法，省力省时间。（程

知乎周刊-编程小白学 ... 期）
免费领取

序员所说的「轮子」是什么东西？创造它有什么意义？ - 知乎http://www.zhihu.com/question/21818673）今天任何软件工程师都会有意无意地使用很多轮子，从操作系统编译器数据库网络到算法数据结构。想高效地借用轮子，就需要将问题分解再分解，抽象再抽象。任何一个实用的系统（不包括教科书上的示例程序和简单的脚本程序）都需要进行大量的分拆和组合。所以系统设计是编程能力里的高级技能，加合理的假设简化问题尤其有难度，此处不展开讨论。高手和新手的区别在于新手往往不知道轮子的适用范围，而高手的手上轮子数量多且熟知各种轮子的差异，所以对不同的问题可以轻松地找到合适的轮子，当实在找不到合适的轮子时可以自己动手改造现有的轮子。平时有时间拆装和改造已有的轮子会对水平提升有较大帮助。当然能知道怎样快速在搜索引擎里搜出轮子也是一种能力。

编程能力是一种需要考虑扩展性的能力。算法竞赛中的很多算法考虑的是单机的内存算法，计算模型经过高度抽象，在实践中机器的模型更为复杂。比如单机的多级结构带来的各种时间空间复杂度的取舍平衡，多机网络中如何能在提高单机性能外进一步优化整体性能。除了在机器端的扩展，在程序员一端的扩展也很重要。

知乎周刊-编程小白学 ... 期）
免费领取

复杂的问题和工程往往意味着团队协同以及更长时间的开发维护，团队分工和设计沟通这里暂且不论。举个容易被忽视的例子，程序中的注释。高手会更在意完整且表达清楚的注释，因为这是写给现在和未来的团队（包括自己和其他成员）看的，直接影响到长期的整体开发维护效率。

编程能力是一种取舍的能力。局部的最优解未必是全局的最优解。如果一个美妙的解决方案需要将完工时间向后推迟一两个月，需要考虑是否先使用平凡方案解决问题，之后再进行优化。当你的工作延后会阻碍别人的工作时尤其如此。发现一个绝妙的优化方案时先想想这个优化是否真的有价值，如果只是系统中很小的部分，那么不要为了追求心理满足而花很多时间放一个漂亮的轮子上去（参考 Amdahl 定律）。

编程能力是一种预见未来的能力。目前的方案有哪些假设和局限性，在何种情形下会遇到问题甚至崩溃。在未来出现问题时问题是否需要重新定义，系统是否需要重新设计，代码是否需要重构或优化等等都需要未雨绸缪。

知乎周刊-编程小白学 ... 期）
免费领取

编程能力是一种工程能力。无它，唯手熟尔。

编程能力是一种解决问题的能力。如果问题没能被很好地解决，知道再多也没用。

编程能力是一种解决问题的能力。如果问题没能被很好地解决，知道再多也没用。

（重要的事情说三遍，重要的事情说三遍，重要的事情说三遍）

2015-08-31

编程能力是一种工程能力。无它，唯手熟尔。

编程能力是一种解决问题的能力。如果问题没能被很好地解决，知道再多也没用。

编程能力是一种解决问题的能力。如果问题没能被很好地解决，知道再多也没用。

（重要的事情说三遍，重要的事情说三遍，重要的事情说三遍）

2015-08-31

编程入门指南

萧井陌, Badger

CoCode | http://cocode.cc/ ：一个让大家学习、成长、相聚并获得乐趣的技术社区

编程入门指南 一群（243545867）；编程入门指南 二群（438379133）

答疑邮箱： xiao.gua@outlook.com (@萧井陌)

文章结构：

@萧井陌 的编程类回答汇总

...

（由于本期知乎周刊面向的是编程小白，所以本文只节选了作者 2015 年 06 月 07 日更新的 1.4 版本原文中的部分内容。想获知「入门」和「附录」部分的更多精彩内容请详见作者的知乎专栏原文：编程入门指南 v1.4 - 萧井陌的专栏 - 知乎专栏http://zhuanlan.zhihu.com/xiao-jing-mo/19959253）

前言

如今编程成为一个越来越重要的「技能」：作为设计师，懂一些编程可能会帮你更好地理解自己的工作内容；作为创业者，技术创始人的身份则会让你的很多工作显得更容易。而作为刚想入门的新手，面对眼前海量的信息，或许根本不知道从哪里开始；入门轻松度过初级材料的学习后，发现学习越来越困难，陡峭的学习曲

线又让你望而却步；你知道如何在页面上打印输出一些文本行，但是你不知道何时该进行一个真正的有用的项目；你不清楚自己还有哪些不知道的东西，你甚至搞不清下一步该学什么。

这篇文章的内容对此不仅会有一些方向性的建议，还会包含一个基础核心向的编程入门导引（进阶求职向的相关内容目前推荐您关注知友 @Michael282694 的专栏文章：编程者加血，客官进来看看呗！| http://zhuanlan.zhihu.com/Michael282694/20166216）。当然，Step by Step 的路线是不现实的，并且每个人都会有自己的特点，所以给出的这个编程入门导引更多是为了引发读者的思考，最终帮助你形成适用于自己的学习路线。几位读者的实例：

非计算机专业，计算机如何入门？ - freetime 的回答 - 知乎http://www.zhihu.com/question/20194473/answer/42941177

我只想成为一名合格的程序员 | http://my.oschina.net/voler/blog/396424?fromerr=lU7He0zm

知乎《编程入门指南 v1.3》思维导图 | http://blog.csdn.net/a910626/article/details/45223657

但要注意：这篇文章是写给那些真心想学编程的人看的——那些憋着一股狠劲儿，一定要做出个什么东西，不学好不罢休的人；而不是那些「听说编程好玩」的人，在我看来，这种人永远都入不了编程的门，更别提做出个像样的东西来了。

「Disclaimer」：虽然整篇文章的基调都是所谓的「Strong Opinions, Weakly Held」，但读者还是应该像怀疑身边所有东西那样怀疑我们所写内容的准确性。若有任何相关疑问欢迎在知乎或技术社区 CoCode 公开讨论。

与指南相关的知乎问答：

编程入门指南中为什么没有 Java？ - 知乎http://www.zhihu.com/question/28958874

编程入门指南中为什么没有 C#？ - 知乎http://www.zhihu.com/question/31756456

心态调整

确定目标

You can code. They cannot. That is pretty damn cool.

– Learn Python The Hard Way | http://learnpythonthehardway.org/book/

在你学习编程之前思考一下你的目标，当你有最终目标时道路会更加清晰。那么，你想要写什么？网站？游戏？iOS 或者 Android 应用？或是你是想自动化完成一些乏味的任务让你有更多的时间看窗外的风景？也许你只是想更具有就业竞争力找个好工作。所有的这些都是有价值的目标，这些目标都是你编程学习推动力的一部分，没有推动力的人，是无法在略显枯燥的漫长学习之旅中走远的。

这段视频也许能给你启发：What Most Schools Don't Teach | http://v.youku.com/v_show/id_XNTIzNzE2NzQ4.html

不要浮躁

Bad programming is easy. Even Dummies can learn it in 21 days. Good programming requires thought, but everyone can do it and everyone can experience the extreme satisfaction that comes with it.

不管是在线下还是线上的书店，满目都是《21 天学通 Java》这种速成书目，它们都承诺在很短一段时间内就让你能够学会相关技术。Matthias Felleisen 在他的著作 How to Design Programs, Second Edition | http://www.ccs.neu.edu/home/matthias/HtDP2e/Draft/index.html一书中明确指出了这种「速成」的趋势并予以了以上的讽刺。

所谓的「捷径」或者说「银弹」是不存在的，智者说过，精通某个东西需要 10 年或 10000 个小时，也就是汉语中的「十年磨一剑」，所以不用着急，功不唐捐。

培养兴趣

Most good programmers do programming not because they expect to get paid or get adulation by the public, but because it is fun to program.

– Linus Torvalds

沉醉于编程，编程更是为了兴趣。兴趣是推动力的不竭源泉，保持这种充满兴趣的感觉，以便于你能将其投入到你的 10 年/10000 小时的编程时间中。编程很有趣，那是探索的喜悦。那是创造的喜悦。看到自己亲手完成的作品显示在屏幕上很有趣；有人为你的代码而惊叹很有趣；有人在公共场合称赞你的产品、邻居使用你的产品以及在媒体上讨论你的产品很有趣。编程应该十分有趣，若并非如此，就

找出导致编程无趣的问题，然后解决之。

开始学习

令人警醒的故事

刚上初中时我便开始了编程学习，很不幸，我读完了好几本当时普遍存在的诸如《21 天精通 C++》这类的垃圾书，当时读完也无大碍，甚至还能写点小程序。但是软件出故障了我不知道为什么，稍显庞大的编程问题无从下手，碰到现有的库做不到的事也只能两手一摊。虽然我每天不停地编码，但我发现自己的编程能力却是提高得如此缓慢，对于「迭代」与「递归」的概念只有极其有限的了解，可以说只是把计算机当成了计算器来使用。

进入大学后，我主修了物理学，最初的一段时间里我一直在记忆背诵那些物理公式，却不理解它们是如何得出的，它们之间有什么联系，抑或是它们的意义。我不停地学习如何计算解答一些常见的物理问题，却对在这些 Hows 背后的 Whys

一无所知。

而在我尝试做一些基于物理行为的电脑游戏时我再次遇到了之前的困难：面对新问题时无从下手，面对新问题时的恐惧不断累积滋生，我开始主动逃避，不去真正地理解，而是幻想能通过 Google 搜索复制粘贴代码解决问题。幸运的是，大二时的一堂课完全改变了我的学习方法。那是第一次我有了「开天眼」的感觉，我痛苦地意识到，我对一些学科只有少得可怜的真正的理解，包括我主修的物理与辅修的计算机科学。

关于那堂课：那时我们刚刚学习完电学和狭义相对论的内容，教授在黑板上写下了这两个主题，并画了一根线将它们连了起来。「假设我们有一个电子沿导线以相对论级别的速度移动……」，一开始教授只是写下了那些我们所熟悉的电学与狭义相对论的常见公式，但在数个黑板的代数推导后，磁场的公式神奇地出现了。虽然几年前我早已知道这个公式，但那时我根本不知道这些现象间有着这样潜在的联系。磁与电之间的差别只是「观察角度」的问题，我猛然醒悟，此后我不再仅仅追求怎么做(How)，我开始问为什么(Why)，开始回过头来，拾起那些最基础的部分，

学习那些我之前我本该好好学的知识。这个回头的过程是痛苦的，希望你们能就此警醒，永远不要做这种傻事。

警醒后的反思

这幅图取自 Douglas Hofstadter 的著作 Gödel, Escher, Bach。图中的每一

个字母都由其他更小的字母组成。在最高层级，我们看的是"MU"，M 这个字母由三个 HOLISM（整全观）构成，U 则是由一个 REDUCTIONISM（还原论）构成，前者的每一个字母都包含后者的整个词，反之亦然。而在最低层级，你会发现最小的字母又是由重复的"MU"组成的。

每一层次的抽象都蕴含着信息，如果你只是幼稚地单一运用整体论在最高层级观察，或运用还原论观察最低层级，你所得到的只有"MU"（在一些地区的方言中 mu 意味着什么都没有）。问题来了，怎样才能尽可能多地获取每个层级的信息？或者换句话说，该怎样学习复杂领域（诸如编程）包含的众多知识？

教育与学习过程中普遍存在一个关键问题：初学者们的目标经常过于倾向整全观而忽略了基础，举个常见的例子，学生们非常想做一个机器人，却对背后的

理解物理模型 → 理解电子工程基础 → 理解伺服系统与传感器 → 让机器人动起来

退出阅读
专题编程小白学 Python 45%

这一过程完全提不起兴趣。

在这里对于初学者有两个大坑：

1. 如果初学者们只与预先构建好的「发动机和组件」接触（没有理解和思考它们构造的原理），这会严重限制他们在将来构建这些东西的能力，并且在诊断解决问题时无从下手。

2. 第二个坑没有第一个那么明显：幼稚的「整体论」方法有些时候会显得很有效，这有一定的隐蔽性与误导性，但是一两年过后（也许没那么长），当你在学习路上走远时，再想回过头来「补足基础」会有巨大的心理障碍，你得抛弃之前自己狭隘的观念，耐心地缓步前进，这比你初学时学习基础知识困难得多。

但也不能矫枉过正，陷入还原论的大坑，初学时便一心试图做宏大的理论，这样不仅有一切流于理论的危险，枯燥和乏味还会让你失去推动力。这种情况经常发生在计算机科班生身上。

退出阅读
专题编程小白学 Python 45%

为了更好理解，可以将学习编程类比为学习厨艺：你为了烧得一手好菜买了一些关于菜谱的书，如果你只是想为家人做菜，这会是一个不错的主意，你重复菜谱上的步骤也能做出不赖的菜肴，但是如果你有更大的野心，真的想在朋友面前露一手，做一些独一无二的美味佳肴，甚至成为「大厨」，你必须理解这些菜谱背后大师的想法，理解其中的理论，而不仅仅是一味地实践。但是如果你每天唯一的工作就是阅读那些厚重的理论书籍，因为缺乏实践，你只会成为一个糟糕的厨子，甚至永远成为不了厨子，因为看了几天书后你就因为枯燥放弃了厨艺的学习。

总之，编程是连接理论与实践的纽带，是计算机科学与计算机应用技术相交融的领域。正确的编程学习方法应该是：通过自顶而下的探索与项目实践，获得编程直觉与推动力；从自底向上的打基础过程中，获得最重要的通用方法并巩固编程思想的理解。

作为初学者，应以后者为主，前者为辅。

退出阅读
专题编程小白学 Python 46%

启蒙

「学编程应该学哪门语言？」这经常是初学者问的第一个问题，但这是一个错误的问题，你最先考虑的问题应该是「哪些东西构成了编程学习的基础」？

编程知识的金字塔底部有三个关键的部分：

算法思想：例如怎样找出一组数中最大的那个数？首先你得有一个 maxSoFar 变量，之后对于每个数……
语法：我怎样用某种编程语言表达这些算法，让计算机能够理解。
系统基础：为什么 while(1) 时线程永远无法结束？为什么 int *foo() { int x = 0; return &x; } 是不可行的？

启蒙阶段的初学者若选择 C 语言作为第一门语言会很困难并且枯燥，这是因为他们被迫要同时学习这三个部分，在能做出东西前要花费很多时间。

因此，为了尽量最小化「语法」与「系统基础」这两部分，建议使用 Python 作为学习的第一门语言，虽然 Python 对初学者很友好，但这并不意味着它只是一个「玩具」，在大型项目中你也能见到它强大而灵活的身影。熟悉 Python 后，学习 C 语言是便是一个不错的选择了：学习 C 语言会帮助你以靠近底层的视角思考问题，并且在后期帮助你理解操作系统层级的一些原理，如果你只想成为一个普通（平庸）的开发者你可以不学习它。

下面给出了一个可供参考的启蒙阶段导引，完成后你会在头脑中构建起一个整体框架，帮助你进行自顶向下的探索。

1. 完成 Learn Python The Hard Way | http://learnpythonthehardway.org/book/或者《「笨办法」学 Python（第 3 版）》| http://book.douban.com/subject/26264642/

2. 完成 MIT 计算机导论课 | https://www.edx.org/course/introduction-co

mputer-science-mitx-6-00-1x-6#.VNL-zlWUdQ0（如果你英语不过关：麻省理工学院公开课：计算机科学及编程导论 | http://www.xuetangx.com/courses/MITx/6_00_1x/2014_T2/about）。MOOC 是学习编程的一个有效途径。虽然该课程的教学语言为 Python，但作为一门优秀的导论课，它强调学习计算机科学领域里的重要概念和范式，而不仅仅是教你特定的语言。如果你不是科班生，这能让你在自学时开阔眼界；课程内容：计算概念，Python 编程语言，一些简单的数据结构与算法，测试与调试。支线任务：完成《Python 核心编程》| http://book.douban.com/subject/3112503/

3. 完成 Harvard CS50 | https://www.edx.org/course/introduction-computer-science-harvardx-cs50x#.VNyhfFWUdQ1（如果你英语不过关：完成哈佛大学公开课：计算机科学 cs50 | http://v.163.com/special/opencourse/cs50.html）。同样是导论课，但这门课与 MIT 的导论课互补。教学语言涉及 C, PHP, JavaScript + SQL, HTML + CSS，内容的广度与深度十分合理，还能够了解到最新的一些科技成果，可以很好激发学习计算机的兴趣。支线任务：

阅读《编码的奥秘》| http://book.douban.com/subject/1024570/
完成《C 语言编程》| http://book.douban.com/subject/1786294/
[可选] 如果你的目标是成为一名 Hacker：阅读 Hacker's Delight | http://book.douban.com/subject/1784887/

PS：如果教育对象还是一个孩子，以下的资源会很有帮助（年龄供参考）：

5-8 岁： Turtle Academy | http://turtleacademy.com/

8-12 岁：Python for Kids | http://jasonrbriggs.com/python-for-kids/

12 岁以上： MIT Scratch | http://scratch.mit.edu/（不要小看 Scratch，有人用它写 3D 渲染的光线追踪系统）或 Khan Academy | https://www.khanacademy.org/computing/computer-programming

入门

结束启蒙阶段后，初学者积累了一定的代码量，对编程也有了一定的了解。这时你可能想去学一门具体的技术，诸如 Web 开发，Android 开发，iOS 开发什么的，你可以去尝试做一些尽可能简单的东西，给自己一些正反馈，补充自己的推动力。但记住别深入，这些技术有无数的细节，将来会有时间去学习；同样的，这时候也别过于深入特定的框架和语言，现在是学习计算机科学通用基础知识的时候，不要试图去抄近路直接学你现在想学的东西，这是注定会失败的。

那么入门阶段具体该做些什么呢？这时候你需要做的是反思自己曾经写过的程序，去思考程序为什么(Why)要这样设计？，思考怎样(How)写出更好的程序？试图去探寻理解编程的本质：利用计算机解决问题。

设想 ：

X = 用于思考解决方案的时间，即「解决问题」 部分

Y = 用于实现代码的时间，即「利用计算机」部分」

编程能力 = F(X, Y)（X>Y）

要想提高编程能力，就得优化 X，Y 与函数 F(X, Y)，很少有书的内容能同时着重集中在这三点上，但有一本书做到了——Structure and Interpretation of Computer Programs(SICP)《计算机程序的构造和解释》| http://mitpress.mit.edu/sicp/full-text/book/book.html，它为你指明了这三个变量的方向。在阅读 SICP 之前，你也许能通过调用几个函数解决一个简单问题。但阅读完 SICP 之后，你会学会如何将问题抽象并且分解，从而处理更复杂更庞大的问题，这是编程能力巨大的飞跃，这会在本质上改变你思考问题以及用代码解决问题的方式。此外，SICP 的教学语言为 Scheme，可以让你初步了解函数式编程。更重要的是，他的语法十分简单，你可以很快学会它，从而把更多的时间用于学习书中的编程思想以及复杂问题的解决之道上。

Peter Norvig 曾经写过一篇非常精彩的 SICP书评 | http://www.amazon.com/review/R403HR4VL71K8/ref=cm_cr_rdp_perm，其中有这样一段：

To use an analogy, if SICP were about automobiles, it would be for the person who wants to know how cars work, how they are built, and how one might design fuel-efficient, safe, reliable vehicles for the 21st century. The people who hate SICP are the ones who just want to know how to drive their car on the highway, just like everyone else.

如果你是文中的前者，阅读 SICP 将成为你衔接启蒙与入门阶段的关键点

虽然 SICP 是一本「入门书」，但对于初学者还是有一定的难度。

其他

编程入门阶段比较容易忽视的几点：

1. 学好英语：英语是你获取高质量学习资源的主要工具，但在入门阶段，所看的那些翻译书信息损耗也没那么严重，以你自己情况权衡吧。此外英语的重要性更

体现在沟通交流上，Linus Torvalds 一个芬兰人，一口流利的英语一直是他招募开发者为 Linux 干活的法宝，这是你的榜样。

2. 学会提问：学习中肯定会遇到问题，首先应该学会搜索引擎的「高级搜索」，当单靠检索无法解决问题时，去 Stack Overflow | http://stackoverflow.com/或知乎 | http://www.zhihu.com/提问，提问前读读这篇文章：What have you tried? | http://mattgemmell.com/what-have-you-tried/

3. 不要做一匹独狼：尝试搭建一个像http://ezyang.com/这样简单的个人网站，不要只是一个孤零零的 About 页面，去学习 Markdown 与 LaTeX，试着在 Blog 上记录自己的想法，并订阅自己喜欢的编程类博客。推荐几个供你参考：

Joel on Software | http://www.joelonsoftware.com/,

Peter Norvig | http://www.norvig.com/index.html ,

Coding Horror | http://blog.codinghorror.com/

小结

以上的内容你不应该感到惧怕，编程的入门不是几个星期就能完成的小项目。期间你还会遇到无数的困难，当你碰壁时试着尝试「费曼」技巧：将难点分而化之，切成小知识块，再逐个对付，之后通过向别人清楚地解说来检验自己是否真的理解。当然，依旧会有你解决不了的问题，这时候不要强迫自己——很多时候当你之后回过头来再看这个问题时，一切豁然开朗。

此外不要局限于上文提到的那些材料，还有一些值得在入门阶段以及将来的提升阶段反复阅读的书籍。这里不得不提到在 stackoverflow 上票选得出的程序员必读书单中，排在前两位的两本书：

Code Complete | http://book.douban.com/subject/1477390/：不管是对于经验丰富的程序员还是对于那些没有受过太多的正规训练的新手程序员，此书都

能用来填补自己的知识缺陷。对于入门阶段的新手们，可以重点看看涉及变量名，测试，个人性格的章节。

The Pragmatic Programmer | http://book.douban.com/subject/1417047/: 程序员入门书，终极书。有人称这本书为代码小全：从 DRY 到 KISS，从做人到做程序员，这本书教给了你一切，你所需的只是遵循书上的指导。

这本书的作者 Dave ，在书中开篇留了这样一段话：

You’re a Pragmatic Programmer. You aren’t wedded to any particular technology, but you have a broad enough background in the science, and your experience with practical projects allows you to choose good solutions in particular situations. Theory and practice combine to make you strong. You adjust your approach to suit the current circumstances and environment. And you do this continuously as the work progresses. Pragmatic Programmers get the job done, and do it well.

这段话以及他创立的 The Pragmatic Bookshelf | https://pragprog.com/一直以来都积极地影响着我，因此这篇指南我也尽量贯彻了这个思想，引导并希望你们成为一名真正的 Pragmatic Programmer 。

后记

如果你能设法完成以上的所有任务，恭喜你，你已经真正实现了编程入门。这意味着你在之后更深入的学习中，不会畏惧那些学习新语言的任务，不会畏惧那些「复杂」的 API，更不会畏惧学习具体的技术，甚至感觉很容易。当然，为了掌握这些东西你依旧需要大量的练习，腰还是会疼，走路还是会费劲，一口气也上不了 5 楼。但我能保证你会在思想上有巨大的转变，获得极大的自信，看老师同学和 CSDN 的眼光会变得非常微妙，虽然只是完成了编程入门，但已经成为了程序员精神世界的高富帅。不，我说错了，即使是高富帅也不会有强力精神力，他也会怀疑自己，觉得自己没钱就什么都不是了。但总之，你遵循指南好好看书，那就会体验「会当凌绝顶」的感觉。

欢迎实践过的同学到知乎上来现身说法。