今天给各位分享从零开始 人工智能的知识，其中也会对从零开始人工智能进行解释，如果能碰巧解决你现在面临的问题，别忘了关注本站，现在开始吧！

本文目录

如何从零开始学编程

Youcancode.Theycannot.Thatisprettydamncool.–LearnPythonTheHardWay

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的清晰。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。

不要浮躁

Badprogrammingiseasy.EvenDummiescanlearnitin21days.Goodprogrammingrequiresthought,buteveryonecandoitandeveryonecanexperiencetheextremesatisfactionthatcomeswithit.

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。MatthiasFelleisen在他的著作HowtoDesignPrograms,SecondEdition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

Mostgoodprogrammersdoprogrammingnotbecausetheyexpecttogetpaidorgetadulationbythepublic,butbecauseitisfuntoprogram.

–LinusTorvalds

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。

这幅图取自DouglasHofstadter的著作G?del,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（整全觀）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后者整个词，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，怎样才能尽可能多的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觀而忽略了基础，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

这一过程完全提不起兴趣。

在这里对于初学者有两个大坑：

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。

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。

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。

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(){intx=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。

完成Codecademy的Python部分。这只是热身部分，尽快完成它，因为你永远只是在浏览器里，你不会学到如何搭建开发环境。在Codecademy这类的编程学习网站学到的那点儿东西，哪怕你只想做一个小的不能再小的项目，你都不知道该从哪儿开始。

完成MIT6.00.1x（中文化）（如果你英语不过关，完成麻省理工学院公开课：计算机科学及编程导论。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

完成HarvardCS50(如果你英语不过关：完成哈佛大学公开课：计算机科学cs50。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》

完成《C语言编程》

[可选]如果你的目标是成为一名Hacker：阅读Hacker'sDelight

PS：如果教育对象还是一个孩子，以下的资源会很有帮助：

5-8岁：TurtleAcademy

8-12岁：PythonforKids

12岁以上：MITScratch或KhanAcademy

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——StructureandInterpretationofComputerPrograms(SICP)《计算机程序的构造和解释》，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。

PeterNorvig曾经写过一篇非常精彩的SICP书评，其中有这样一段：

Touseananalogy,ifSICPwereaboutautomobiles,itwouldbeforthepersonwhowantstoknowhowcarswork,howtheyarebuilt,andhowonemightdesignfuel-efficient,safe,reliablevehiclesforthe21stcentury.ThepeoplewhohateSICParetheoneswhojustwanttoknowhowtodrivetheircaronthehighway,justlikeeveryoneelse.

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，以下是一些十分有用的辅助资源：

UdacityCS212DesignofComputerProgram)：由上文提到的Google研究主管PeterNorvig主讲，教学语言为Python，内容有一定难度。

HowtoDesignPrograms,SecondEdition：HtDP的起点比SICP低，书中的内容循循善诱，对初学者很友好，如果觉得完成SICP过于困难，可以考虑先读一读HtDP。

UCBerkeleySICP授课视频以及SICP的两位作者给Hewlett-Packard公司员工培训时的录像(中文化项目)

ComposingPrograms：一个继承了SICP思想但使用Python作为教学语言的编程导论（其中包含了一些小项目）

SICP解题集：对于书后的习题，作为初学者应尽力并量力完成。

完成了这部分学习后，你会逐步建立起一个自己的程序设计模型，你的脑子里不再是一团乱麻，你会意识到记住库和语法并不会教你如何解决编程问题，接下来要学些什么，在你心里也会明朗了很多。这时候才是真正开始进行项目实践，补充推动力的好时机。

关于项目实践：对于入门阶段的初学者，参与开源项目还为时过早，这时候应该开始一些简单的项目，诸如搭建一个网站并维护它，或是编写一个小游戏再不断进行扩展，如果你自己的想法不明确，MegaProjectList中选取项目。总之，务必在这时拿下你项目实践的第一滴血。

与此同时，别忘了继续打好根基。为了将来的厚积薄发，在下面这几个方面你还要继续做足功课（注意：下面的内容没有绝对意义上的先后顺序）：

计算机系统基础

有了之前程序设计的基础后，想更加深入地把握计算机科学的脉络，不妨看看这本书：《深入理解计算机系统》ComputerSystemsAProgrammer'sPerspective。这里点名批评这本书的中译名，其实根本谈不上什么深入啦，这本书只是CMU的「计算机系统导论」的教材而已。CMU的计算机科学专业相对较偏软件，该书就是从一个程序员的视角观察计算机系统，以「程序在计算机中如何执行」为主线，全面阐述计算机系统内部实现的诸多细节。

如果你看书觉得有些枯燥的话，可以跟一门Coursera上的MOOC:TheHardware/SoftwareInterface，这门课的内容是CSAPP的一个子集，但是最经典的实验部分都移植过来了。同时，可以看看TheCProgrammingLanguage，回顾一下C语言的知识。

完成这本书后，你会具备坚实的系统基础，也具有了学习操作系统，编译器，计算机网络等内容的先决条件。当学习更高级的系统内容时，翻阅一下此书的相应章节，同时编程实现其中的例子，一定会对书本上的理论具有更加感性的认识，真正做到经手的代码，从上层设计到底层实现都了然于胸，并能在脑中回放数据在网络->内存->缓存->CPU的流向。

此外，也是时候去接触UNIX哲学了:KISS-KeepitSimple,Stupid.在实践中，这意味着你要开始熟悉命令行界面，配置文件。并且在开发中逐渐脱离之前使用的IDE，学会使用Vim或Emacs（或者最好两者都去尝试）。

阅读《UNIX编程环境》

阅读《UNIX编程艺术》

折腾你的UN*X系统

数据结构与算法基础

如今，很多人认为编程（特别是做web开发）的主要部分就是使用别人的代码，能够用清晰简明的方式表达自己的想法比掌握硬核的数学与算法技巧重要的多，数据结构排序函数二分搜索这不都内置了吗？工作中永远用不到，学算法有啥用啊？这种扛着实用主义大旗的「码农」思想当然不可取。没有扎实的理论背景，遭遇瓶颈是迟早的事。

数据结构和算法是配套的，入门阶段你应该掌握的主要内容应该是：这个问题用什么算法和数据结构能更快解决。这就要求你对常见的数据结构和算法了熟于心，你不一定要敲代码，用纸手写流程是更快的方式。对你不懂的数据结构和算法，你要去搜它主要拿来干嘛的，使用场景是什么。

供你参考的学习资源：

《算法导论》：有人说别把这本书当入门书，这本书本来就不是入门书嘛，虽说书名是IntroductiontoAlgorithms，这只不过是因为作者不想把这本书与其他书搞重名罢了。当然，也不是没办法拿此书入门，读第一遍的时候跳过习题和证明就行了

嘛，如果还觉得心虚先看看这本《数据结构与算法分析》

CourseraAlgorithms:DesignandAnalysis[Part1]&[Part2]：Stanford开的算法课，不限定语言，两个部分跟下来算法基础基本就有了；英语没过关的：麻省理工学院公开课：算法导论

入门阶段还要注意培养使用常规算法解决小规模问题的能力，结合前文的SICP部分可以读读这几本书：《编程珠玑》，《程序设计实践》

编程语言基础

Differentlanguagessolvethesameproblemsindifferentways.Bylearningseveraldifferentapproaches,youcanhelpbroadenyourthinkingandavoidgettingstuckinarut.Additionally,learningmanylanguagesisfareasiernow,thankstothewealthoffreelyavailablesoftwareontheInternet

-ThePragmaticProgrammer

此外还要知道，学习第n门编程语言的难度是第(n-1)门的一半，所以尽量去尝试不同的编程语言与编程范式，若你跟寻了前文的指引，你已经接触了：「干净」的脚本语言Python,传统的命令式语言C,以及浪漫的函数式语言Scheme/Racket三个好朋友。但仅仅是接触远远不够，你还需要不断继续加深与他们的友谊，并尝试结交新朋友，美而雅的Ruby小姑娘，Hindley-Milner语言家族的掌中宝Haskell都是不错的选择。但有这么一位你躲不开的，必须得认识的大伙伴—C++，你得做好与他深交的准备：

入门：C++Primer

[可选]进阶：

高效使用：EffectiveC++

深入了解：《深度探索C++对象模型》；C++Templates

研究反思：TheDesignandEvolutionofC++；对于C++这个NecessaryEvil，看这本书可以让你选择是成为守夜人还是守日人。

现实是残酷的，在软件工程领域仍旧充斥着一些狂热者，他们只掌握着一种编程语言，也只想掌握一种语言，他们认为自己掌握的这门语言是最好的，其他异端都是

傻X。这种人也不是无药可救，有一种很简单的治疗方法：让他们写一个编译器。要想真正理解编程语言，你必须亲自实现一个。现在是入门阶段，不要求你去上一门编译器课程，但要求你能至少实现一个简单的解释器。

供你参考的学习资源：

《程序设计语言-实践之路》：CMU编程语言原理的教材，程序语言入门书，现在就可以看，会极大扩展你的眼界，拉开你与普通人的差距。

Coursera编程语言MOOC：课堂上你能接触到极端FP（函数式）的SML，中性偏FP的Racket，以及极端OOP（面向对象）的Ruby，并学会问题的FP分解vsOOP分解、ML的模式匹配、Lisp宏、不变性与可变性、解释器的实现原理等，让你在将来学习新语言时更加轻松并写出更好的程序。

UdacityCS262ProgrammingLanguage：热热身，教你写一个简单的浏览器——其实就是一个javascript和html的解释器，完成后的成品还是很有趣的；接下来，试着完成一个之前在SICP部分提到过的项目：用Python写一个SchemeInterpreter

其他

编程入门阶段比较容易忽视的几点：

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，LinusTorvalds一个芬兰人，一口流利的英语一直是他招募开发者为Linux干活的的法宝，这是你的榜样。

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StackOverflow或知乎提问，提问前读读这篇文章：Whathaveyoutried?

不要做一匹独狼：尝试搭建一个像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：JoelonSoftware,PeterNorvig,CodingHorror

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。

此外不要局限与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。ThePragmaticProgrammer就是这样一本程序员入门书，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。

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。

学技术，人工智能怎么样？

智能化的时代学这个还是很不错的，毕竟现在科技这么发达，不断更新，时代的一个进步，社会的进步，以后的生活基本都离不开智能化！

如何从零开始学习AI软件

1、对于精通PS的设计师来说，AI有很多相似之处，学起来更加容易，如果PS不熟练，可以先买本书阅读下基本的理论知识，了解AI的界面和工具选项栏的作用。推荐电子书和纸质书。

2、大概熟悉之后，在电脑要安装AI软件，打开软件，进行最基本的操作，所谓熟能生巧，多练多看，达到很熟悉的程度。

3、学会使用快捷键，也可以自己设置快捷方式，快捷键可以帮助我们提高工作效率，还有就是掌握一些操作技巧，这些能够提高我们的速度和更加理解工具的应用

。

4、简单模仿，看一些简单的素材文件，开始模仿其操作，想像一下要怎么实现操作，应用了哪些工具。

5、自己定义目标，根据创作理念，开始发挥创作性思维，用学到的知识填补画面，设计一副完整的作品。

6、最重要的还是要多看大师们的作品，领悟其精髓，化为已用，多看多思考，形成自己的设计风格。

扩展

Adobeillustrator，常被称为“AI”，是一种应用于出版、多媒体和在线图像的工业标准矢量插画的软件。

作为一款非常好的矢量图形处理工具，该软件主要应用于印刷出版、海报书籍排版、专业插画、多媒体图像处理和互联网页面的制作等，也可以为线稿提供较高的精度和控制，适合生产任何小型设计到大型的复杂项目。

人工智能主要是学什么的？

什么是人工智能

人工智能，英文名为ArtificialIntelligence，也就是人们口中的AI。它是研究、开发用于模拟、延伸和扩展人的智能的理论、方法、技术及应用系统的一门新技术科学。

人工智能的主旨是研究和开发出智能实体，??在这一点上它属于工程学。由于覆盖的领域非常广泛，所以??这是一门??集众多学科精华的??尖端技术。？

人工智能主要学什么

人工智能涉及的学科包括：计算机科学、信息论、控制论、自动化、仿生学、生物学、医学等多门学科。

目前国内高校本科生阶段的专业目录中并没有设置人工智能专业，在研究生阶段才开设相应的研究方向。但是本科阶段有很多专业是与人工智能相关的，比如计算机类、电子信息类、自动化类、数学类。

计算机类包含：计算机科学与技术、软件工程、网络工程、信息安全、物联网工程、集成电路设计与集成系统等学科

自动化类包含：自动化、轨道交通信号与控制等

数学类包含：数学与应用数学、信息与计算科学、数理基础科学、数据科学与大数据技术等

人工智能的前景如何

作为新一轮产业变革的核心驱动力和引领未来发展的战略技术，国家高度重视人工智能产业的发展。

尤其是5G概念提出并实现后，人工智能的发展更是一片光明，自动驾驶和智慧城市、智能家具等等人工智能产品逐渐走进人们的视线，试想一下，在未来的某一天，电影中的黑科技生活全都成为现实，科技感爆棚！这即是人类发展进步的一大标志，更是未来的又一新机遇。

更多优质内容，请持续关注镁客网~

文章分享结束，从零开始
人工智能和从零开始人工智能的答案你都知道了吗？欢迎再次光临本站哦！