计算机科学基础班（第三期）报名

（第三期课程已经报名结束，并且开始教学。有意报名第四期的同学可以发送申请，我可以提前进行面试工作，不过需要等两个月以后才能开始了。也可以考虑最近开始研制的阅读班。）

计算机科学基础班（第二期）已经成功结束两个月了。每次的教学都让我发现以前没有注意到的细节，以至于每一次都在改进。现在休息了两个月之后，我觉得大概可以召集第三期的报名工作了。

第二期课程总结

第二期课程调低了学费，增加了课程规模。虽然减轻了学生的经济负担，让更多的人能够参加，总体的效果也很好，但这使得我和助教的工作都比较辛苦。

我的教学跟普通学校有很大的不同，不仅在于内容，讲课方式，还在于精心设计的练习。练习是教学中很重要的部分，学生自身能力的提高，其实主要是由循序渐进的练习来完成的。就像设计良好的健身练习一样，它们会逐渐让学生的头脑变得强壮，而不是让他们半途放弃或者受伤。

每一个练习都是单独提交，而不是都做完才一次性提交。这样学生会得到准确而及时的反馈，避免重复犯错误。对于思路不清而卡住的练习，也会收到量身定制的准确提示，让学生意识到错误所在，思维走上正路，却不“剧透”最终结果。这样的设计，使得学生的思维得到最大限度的锻炼，逐渐获得独立思考的能力。

对于学生的练习，我的要求不仅是要正确，而且要极度简单，没有任何多余的东西，逻辑严密清晰，就算空白和排版都要符合最高的标准。每一个练习都可能被要求多次修改之后才能通过。从最小的程序出发，学生从一开头就养成逻辑严密的习惯，直到复杂的代码都保持这种习惯，所以程序极少出现错误。从这个课程，学生会自然地掌握我在《编程的智慧》一文里指出的各种方面。几乎没有任何大学会纠正和引导学生这方面的风格，而这其实是很重要的。这就是为什么这么多人博士毕业了，却做出那么复杂而容易出错的设计来。

我为每个学生都建立了一个“辅导群”，里面有四个人：学生，老师和两个助教。这样的设计使得学生能最大限度的得到反馈。即使老师在忙其他事，助教有时间也可以回复。助教不清楚的地方，都由老师亲自来看。所以虽然是集体班，但其实每个人都得到了近似于一对一的教学。

相比大学里的情况，教授和助教是只有上课时或 office hours 才见

得到的。作业都是一次性提交批改，不可能来回的提示和指引。这种特别的教学方式，使得对时间的利用效率大大提高。课后的每个练习本身也都变成了教学，所以虽然只有 8 节课，实际的教学时间却是大大高于讲课时间的。大部分时间是学生在用功，老师只需要在关键时刻点一下就行。

做这样的练习是如此有趣，以至于到后面几节课时，学生们都希望我少讲，少“剧透”，从而可以把更多的知识点作为练习给他们自己思考，以至于中间有一两节课几乎没讲什么内容，大部分作为练习发放了。这样的教学方式和效果是我的生涯中前所未见的，可能是世界上独一无二的。我在中国和美国待过四个大学，没有任何一门课程，任何书籍像我讲的这么少，学生却学到这么多。

虽然效果很好，练习的回复也增加了老师和助教的工作。第二期课程中的许多天，我和助教们直到晚上 12 点都还在回复学生的练习，给他们提示和指点。由于很多学生平时要工作，所以直到周末才开始做练习，以至于周末的时候涌来大量的练习信息。我有两个非常认真热心的助教，有时候看到他们深夜和周末还在忙着回复学生，我都叫他们快去休息。助教的反映是，虽然微信聊天都设置为了“免打扰”，学生也知道晚上不期待我们回复，但看到学生提交的练习，总是忍不住会去看，去回复。我可以说，在当今的世界，可能很少有人会这么在乎教学这个事了。

第三期课程报名

因为第二期的学费和学生数量，使得教学进行比较累，所以第三期课程不会再采用第二期的学费价格和班级大小。第三期课程的具体计划如下：

1. 学费调整回原先的 12000 人民币，可以接受相应的美元付款。
2. 班级大小限制为 15 人以内。
3. 讲课课时仍然是 8 节课，每节课大概 2 小时。但因为我发现其中有一两节课内容很少，时间其实没有很好利用，所以也可能把其中的一两节课换成练习发布。
4. 因为讨论会效果不是很明显，而且可能占用学生的工作时间，不再设置每星期一次的讨论会。
5. 由于第二期的两位助教付出了很多辛苦，所以这次课程不再让他们做助教。这次可能没有助教，不过如果有以前表现出色的学生自愿做助教，也可以考虑。
6. 上课时间一般会定在周中某一天晚上 8 点到 10 点。因为涉及到比较多人，上课时间一旦定下之后就不再因为个人的工作时间变动而改动。课程不提供录播，所以加班较多的学生请考虑好自己的工作时间，错过一节课的代价可能会很大。
7. 课程会在春节之后，经过对申请进行筛选，班级容量达到之后

开始。

报名方式

报名请发送 email 到 yinwang.advising@icloud.com（请勿向这个 email 发送其它主题的信件）。标题为《计算机科学基础班（第三期）报名》。来信请说明自己的基本信息，附件发送一个简历。由于班级人数有限，而且为了课程的顺利，会对申请人进行选择。像申请国外大学一样，请写一段“personal statement”说明自己的学习动机。因为你的态度决定了是否提供课程，所以对于申请请慎重，不要着急和轻率，学生的选择不是按照申请的时间顺序的。如果觉得合适，我会通知你进行下一步的面试。

课程大纲

根据第二期课程的经验，我想对课程的内容做一个比以前详细的说明。之前一直对课程内容没有很多说明，一方面为的是留下自由发挥的空间，一方面是为了让学生有一定的神秘感，引发好奇心。但这么简单的说明似乎会让不知情的人误以为“已经学过这些东西”，有时候会发现一些人看了说明之后，自以为我教的内容他都会了。我只为他们感到可惜。

下面简要说一下课程的内容：

教学语言。课程目前使用 JavaScript 作为教学语言，但并不是教 JavaScript 语言本身，不会使用 JavaScript 特有的任何功能。课程教的思想不依赖于 JavaScript 的任何特性，它可以应用于任何语言，课程可以在任何时候换成任何语言。学生从零开始，学会的是计算机科学最核心的思想，从无到有创造出各种重要的概念，直到最后实现出自己的编程语言和类型系统。

课程强度。课程的设计是一个逐渐加大难度，比较辛苦，却很安全的山路，它通往很高的山峰。要参加课程，请做好付出努力的准备。在两个月的时间里，你每天需要至少一个小时来做练习，有的练习需要好几个小时才能做对。跟其他的计算机教学不同，学生不会因为缺少基础而放弃，不会误入歧途，也不会掉进陷阱出不来。学生需要付出很多的时间和努力，但没有努力是白费的。

曾经有一两个学生因为低估了学习的强度，同时又有其他重要任务，结果发现忙不过来，所以请合理的安排，不要在有其他重要任务的同时参加学习。

第一课：函数。跟一般课程不同，我不从所谓“Hello World”程序开始，也不会叫学生做一些好像有趣而其实无聊的小游戏。一开头我就讲最核心的内容：函数。关于函数只有很少几个知识点，但它们却是一切的核心。只知道很少的知识点的时候，对它们进行反复的练习，

让头脑能够自如地对它们进行思考和变换，这是教学的要点。我为每个知识点设计了恰当的练习。

第一课的练习每个都很小，只需要一两行代码，却蕴含了深刻的原理。练习逐渐加大难度，直至超过博士课程的水平。我把术语都改头换面，要求学生不上网搜索相关内容，为的是他们的思维不受任何已有信息的干扰，独立做出这些练习。练习自成系统，一环扣一环。后面的练习需要从前面的练习获得的灵感，却不需要其它基础。有趣的是，经过正确的引导，好些学生把最难的练习都做出来了，完全零基础的学生也能做出绝大部分，这是我在世界名校的学生里都没有看到过的。具体的内容因为不剧透的原因，我就不继续说了。

第二课：递归。递归可以说是计算机科学（或数学）最重要的概念。我从最简单的递归函数开始，引导理解递归的本质，掌握对递归进行系统化思考的思路。递归是一个很多人自以为理解了的概念，而其实很多人都被错误的教学方式误导了。很多人提到递归，只能想起“汉诺塔”或者“八皇后”问题，却不能拿来解决实际问题。很多编程书籍片面强调递归的“缺点”，教学生如何“消除递归”，却看不到问题的真正所在——某些语言（比如 C 语言）早期的函数调用实现是错误而效率低下的，以至于学生被教导要避免递归。由于对于递归从来没有掌握清晰的思路，在将来的工作中一旦遇到复杂点的递归函数就觉得深不可测。

第三课：链表。从零开始，学生不依赖于任何语言的特性，实现最基本的数据结构。第一个数据结构就是链表，学生会在练习中实现许多操作链表的函数。这些函数经过了精心挑选安排，很多是函数式编程语言的基本函数，但通过独立把它们写出来，学生掌握的是递归的系统化思路。这使得他们能自如地对这类数据结构进行思考，解决新的递归问题。

与一般的数据结构课程不同，这个课程实现的大部分都是「函数式数据结构」，它们具有一些特别的，有用的性质。因为它们逻辑结构清晰，比起普通数据结构书籍会更容易理解。与 Haskell 社区的教学方式不同，我不会宗教式的强调纯函数的优点，而是客观地让学生领会到其中的优点，并且发现它们的弱点。学会了这些结构，在将来也容易推广到非函数式的结构，把两种看似不同的风格有机地结合在一起。

第四课：树结构。从链表逐渐推广出更复杂的数据结构——树。在后来的内容中，会常常用到这种结构。树可能是计算机科学中最常用，最重要的数据结构了，所以理解树的各种操作是很重要的。我们的树也都是纯函数式的。

第五课：计算器。在熟悉了树的基本操作之后，实现一个比较高级的计算器，它可以计算任意嵌套的算术表达式。算术表达式是一种“语法树”，从这个练习学生会理解“表达式是一棵树”这样的原理。

第六课：查找结构。理解如何实现 key-value 查找结构，并且亲手实现两种重要的查找数据结构。我们的查找结构也都是函数式数据结构。这些结构会在后来的解释器里派上大的用场，对它们的理解会巩固加深。

第七课：解释器。利用之前打好的基础，亲手实现计算机科学中最重要，也是通常认为最难理解的概念——解释器。解释器是理解各种计算机科学概念的关键，比如编程语言，操作系统，数据库，网络协议，Web 框架。计算机最核心的部件 CPU 其实就是一个解释器，所以解释器的认识能帮助你理解「计算机体系构架」，也就是计算机的“硬件”。你会发现这种硬件其实和软件差别不是很大。你可以认为解释器就是「计算」本身，所以它非常值得研究。对解释器的深入理解，也能帮助理解很多其它学科，比如自然语言，逻辑学。

第八课：类型系统。在解释器的基础上，学生会理解并实现一个相当高级的类型系统（type system）和类型检查器（typechecker）。这相当于实现一个类似 Java 的静态类型语言，但比 Java 在某些方面还要高级和灵活。我们的类型系统包含了对于类型最关键的要素，而不只是照本宣科地讲解某一种类型系统。当你对现有的语言里的类型系统不满意的时候，这些思路可以帮助你设计出自己的类型系统。学生会用动手的方式去理解静态类型系统的原理，其中的规则，却不含有任何公式。

类型系统的规则和实现，一般只会在博士级别的研究中才会出现，可以写成一本厚书（比如 TAPL 那样的），其中有各种神秘的逻辑公式。而我的学生从零开始，一节课就可以掌握这门技术的关键部分，实现出正确的类型系统，并且推导出正确的公式。有些类型规则是如此的微妙，以至于微软这么大的公司在 21 世纪做一个新的语言（TypeScript)，仍然会在初期犯下类型专家们早已熟知的基本错误。上过这个课程的很多同学，可以说对这些基础原理的理解已经超过了 TypeScript 的设计者，但由于接受的方式如此自然，他们有一些人还没有意识到自己的强大。

关于面向对象。虽然课程不会专门讲“面向对象”的思想，但面向对象思想的本质（去掉糟粕）会从一开头就融入到练习里。上过课的同学到后来发现，虽然我从来没直接教过面向对象，而其实他们已经理解了面向对象的本质是什么。在将来的实践中，他们可以用这个思路去看破面向对象思想的本质，并且合理地应用它。

奖励练习。途中我会通过“奖励练习”的方式补充其它内容。比如第二期的课程途中，我临时设计了一个 parser 的练习，做完了其它练习的同学通过这个练习，理解了 parser 的原理，写出了一个简单但逻辑严密的 parser。奖励练习之所以叫“奖励”，因为并不是所有学生都能得到这个练习，只有那些付出了努力，在其他练习中做到融会贯通，学有余力的学生才会给这个练习。这样会鼓励学生更加努力地学习。

如果理解了以上内容蕴含了什么，你可能就不会再问“这些我都学过了，我可不可以参加高级班”了，因为极少有人真的理解了以上内容。就算世界上最高级职位的一些程序员，大学里的教授，对于这些也有很多含糊不清的地方。我自己也通过讲授这些内容得到了启发。

一个朋友看了我的课程内容说，这不叫“基础班”，只能叫“大师班”。他不相信零基础的学生能跟上，但事实却是可行的。为什么不能即是“基础班”又是“大师班”呢？有句话说得好，大师只不过是把基础的东西理解得很透彻的人而已。我希望这个基础班能帮助人们获得本质的原理，帮助他们看透很多其它内容。所以上了“基础班”，可能在很长时间之内都不需要“高级班”了，因为他们已经获得了很强的自学能力，能够自己去探索未知的世界，攀登更高的山峰。