二零一六年关键词————“好的习惯，好的思考，好的表达，好的交流”

“在那些年月里，我们为了推翻卓越的前辈们，废寝忘食地不知付出了多大的努力。现在的风尚的确与当时不大相同了。眼下，日本围棋界的条件相当优越。比如学棋的场所、学棋的对手几乎遍地皆是。然而只有这些，似乎还不足以激发棋手们学习的冲动，因为能让他们感动的东西太少了。或许正是因为年轻棋手们拥有的太多，结果反倒成了他们的不幸。”

题记

“好的习惯”想表达的意思和前两年相比并无改变，只是今年我想确切地形成一两个助益一生的习惯，比如说规律地睡眠和锻炼，保持解题 (编程，数学，游戏，etc) 的胃口。今年我加入了“好的思考”，因为我发现很多时候先有了“好的思考”，才有“好的表达”。“好的思考”常常需要是深入的，不应该浅尝辄止，只看表象。坚持去做“好的思考”，就可能使自己分析判断的能力越来越强。这样，才是真正通往“自己的正解”的路。

补记：到二月份，我意识到还应该有“好的交流”。因为一个人有时尽其所能所能想到做到的事情是很有限的，而找到好的圈子，和好的/强的人交往和沟通能极大的拓展一个人的视野，打破原有的局限。既然意识到了这点，我会试着去寻找好的伙伴好的圈子（比如在老朋友中，在学校，在公司，在网络，通过朋友的朋友，等等），并在其中产生思想的交流和碰撞。

火花卷首语

塞林格写过一部名作叫《麦田里的守望者》，里面的主人公是一个被学校开除的中学生，他貌似玩世不恭，厌倦现存的平庸的一切，但他并非没有理想。他想象悬崖边有一块大麦田，一大群孩子在麦田里玩，而他的理想就是站在麦田边作一个守望者，专门捕捉朝悬崖边上乱跑的孩子，防止他们掉下悬崖。

我很喜欢“守望者”这个名称，它使我想起守林人。守林人的心境总是非常宁静的，他长年与树木、松鼠、啄木鸟这样一些最单纯的生命为伴，他自己的生命也变得单纯了。他的全部生活就是守护森林，瞭望云天，这守望的生涯使他心明眼亮，不染尘嚣。“守望者”的名称还使我想起守灯塔人。在奔流的江河中，守灯塔人日夜守护灯塔，瞭望潮汛，保护着船只的安全航行。当然，与都市人相比，守林人的生活未免冷清。与弄潮儿相比，守灯塔人的工作未免平凡。可是，你决不能说他们是人类中可有可无的一员。如果没有这些守望者的默默守望，森林消失，地球化为沙漠，都市人到哪里去寻欢作乐，灯塔熄灭，航道变为墓穴，弄潮儿如何还能大出风头？

人做事情，或出于利益，或出于性情。凡出于性情做的事情，大都仅仅是为了满足心灵需求，如写作、艺术欣赏、交友等等。我们做这个主页也是如此，愉快是最基本的标准。如果什么时候我们在这里找不到愉快，就必须怀

疑是否有利益的强制在其中起着作用，使性情生活蜕变成了功利行为。火花是这样一个地方，它不想在踌躇满志的游戏精英网站中挤自己的一块地盘，而是很安静的，与世无争的，但也因此在普遍的热闹和竞争中有了存在的价值。我们只想开辟一块园地，让火炎玩家们找到回家的感觉。

如果一个人有自己的心灵追求，又在世界上闯荡了一番，有了相当的人生阅历，那么，他就会逐渐意识到自己在这个世界上的位置。世界无限广阔，诱惑永无止境，然而，属于每一个人的现实可能性终究是有限的。你不妨对一切可能性保持着开放的心态，因为那是人生魅力的源泉，但同时你也要早一些在世界之海上抛下自己的锚，找到最合适自己的领域。一个人不论伟大还是平凡，只要他顺应自己的天性，找到了自己真正喜欢做的事，并且一心把自己喜欢做的事做到尽善尽美，他在这个世界上就有了牢不可破的家园。于是，他不但会有足够的勇气去承受外界的压力，并且会有足够的清醒来面对形形色色的机会的诱惑。我们当然没有理由怀疑，这样的一个人必能获得生活的充实和心灵的宁静。

Jan

1.2 结束 Boston 之行

短短 11 天，没想到发生了那么多事。无论在 **做事**和 **感情**上都有很多体会和收获，同时也让我明确了今年的一些目标和打算。我会好好吸收这些收获和体会，变得更好。

1.3

programming tips

”最开始需要多花时间敲代码，这段时间免不了会敲很多重复的代码，但是无所谓，因为这段时间自己就是一个白痴。然而，当代码可以畅通无阻的表达自己思想的时候，就需要多看书了，让你的思想得到更大的提升，从而可以写出更多更有深度与更有意义的代码。”

《倚天屠龙记》中赵敏是怎么把张无忌追到手的？

赵敏能追到张无忌的秘诀：“我偏要勉强”

常言道：男追女，隔座山；女追男，隔层纱，但是今天讲的“女追男”思路对广大光棍男依然是适用的，因为“女追男、隔层纱”那得看对手是谁。敏敏的对手不是一个，至少是仨！而且除了珠儿，其余都是重量级人物！

小四是金庸作品中女仆系列的代表人物、万千宅男的 YY 对象、人气比肩双儿的小昭！

小三更厉害了——周芷若，听名字就是美女，实际更是美女，貌若天仙，曾经秒杀张无忌、宋青书。同时，她还智慧过人、出身名门，手握武林秘籍，总之，基本上是完美的。放到任何一本武侠小说里做女一号都绝对是绰绰有余的，加上本来就和无忌哥哥熟识多年，从哪个角度看，都是当仁不让的绝对胜利者！

可是，她败了！她不是输给了赵敏，打败她的是爱情!

全书写到一半赵敏才出场，金先生，您还能再 NB 一点么？但是敏敏只用了不到全书八分之一的篇幅就气场全开，征服了几乎所有读者，金先生，你还真能再 NB 了。

敏敏特穆尔，汉语名字为赵敏，身份为蒙古郡主，司职国家安全局，主要职能为剿灭江湖反动人士，比如明教恐怖组织头目张无忌。此女智慧过人，设计将六大派一网打尽，张无忌的各位师叔伯全部落入她手，殷六侠差点丧命，无忌最尊敬的三丰太师傅也差点中招；性格强硬，多次明算、暗算张无忌，屡次弄得张无忌灰头土脸，还差点自杀以谢天下……说到这里，我都不相信她能拖下根正苗红的周芷若成功上位。

她就两招：疯狂爱和水银泻地般犀利无比的表白。前者为体，后者为用，双剑合璧，天下莫当。

疯狂爱，大家都有体会。她放弃郡主身份，放弃大好前程，甚至放弃家庭，义无反顾地去追一个一直站在自己对立面的土匪头子。在张无忌和周芷若结婚典礼上，周围全部是赵敏死敌，她单刀赴会抢亲时，发生了如下对话：【范遥眉头一皱，说道：“郡主，世上不如意事十居八九，既已如此，也是勉强不来了。赵敏道：“我偏要勉强。”】然后我就义无反顾、毅然决然地爱上了这个妖女。

疯狂爱到这就不说了，一来大家可以很容易地从书中体会到，二来这个也不具备可操作性。如果你碰到了你的真命天子，那就由不得你不去疯狂；如果你仅仅是感觉合适或者还算般配，你怎么都疯狂不起来。

下面重点学习敏敏的表白。教科书般的表白，分寸的拿捏，情景的设计，心理的捕捉，都华丽得让我不禁感叹：金老爷子当年肯定也是高手啊！

第一次，赵敏和张无忌在大都的一间小酒店里喝酒。

【张无忌拿起酒杯，火锅的炭火光下见杯边留着淡淡的胭脂唇印，鼻中闻到一阵清幽的香气，也不知这香气是从杯上的唇印而来，还是从她身上而来，不禁心中一荡，便把酒喝了。赵敏道：“再喝两杯。我知道你对我终是不放心，每一杯我都先尝一口。】

深夜，小店，烛光，幽香……这气氛营造得！然后打着反毒品的名义和张无忌同饮一杯酒。要知道，共享饮料的行为一般只发生在关系十分亲密的人之间。此时两人是对立身份，但敏敏反而借势把关系悄然拉近，然后展开一系列人生观、价值观、善恶观的探讨。看见没，不管是追美女还是帅哥，最初都得来点有层次的，谈人生、谈理想是最好、最快、最安全的开始阶段话题。人生理想谈拢以后，敏敏又出招了——【赵敏道：“要是我明天死了，你心里怎样想？你心中一定说：谢天谢地，我这个刁钻凶恶的大对头死了，

从此可免了我不少麻烦。”】摸出张无忌已然对自己有点小动心，立刻以退为进！逼宫！目标达成以后，为了使自己不显得过于强势，立刻又问：【张公子，你说是我美呢，还是周姑娘美?】这叫色诱……洗刷自己强势形象，第一轮较量敏敏完胜无忌，成功把之前的对立形象转化成了无忌心中的美好印象。

然后两人在灵蛇岛“度蜜月”时，赵敏咬了张无忌的手，又在他手上抹去腐消肌膏，这一段，堪称金书调情第一桥段。其后周芷若强势反击，陷害赵敏。赵敏再见无忌之时，已经是杀害殷离、偷走刀剑的嫌疑犯了。【张无忌脸上如罩严霜，喝道：“你要盗那倚天剑和屠龙刀，我不怪你！你将我抛在荒岛之上，我也不怪你！可是殷姑娘已然身受重伤，你何以还要再下毒手！似你这等狠毒的女子，当真天下少见。”说到此处，悲愤难抑，跨上一步，左右开弓，便是四记耳光。赵敏在他掌力笼罩之下，如何闪避得了？啪啪啪啪四声响过，两边脸颊登时红肿。】

如果是你，遭人陷害，九死一生，好不容易见到爱人还不信任自己一顿耳光，你会有什么感受？愤怒？委屈？心冷？不甘？仇恨？那是你们，我们敏敏完全不一样。敏敏此时完全掌握了无忌的心理，看她此时表现：【赵敏追了出来，叫道：“张无忌，你往哪里去?”张无忌道：“跟你有什么相干?”赵敏道：“我有话要问谢大侠和周姑娘，请你带我去见他二人。”张无忌道：“我义父下手不容情，你这不是去送死?”赵敏冷笑道：“你义父心狠手辣，可不似你这等糊涂。再说，谢大侠杀了我，你是报了表妹之仇，岂不是正好偿了你的心愿?”张无忌道：“我糊涂什么？我不愿你去见我义父。”赵敏微笑道：“张无忌，你这糊涂小子，你心中实在舍不得我，不肯让我去给谢大侠杀了，是也不是?”】

后来，张无忌发现殷离是被周芷若杀的，反应却很平淡，连重话都没说两句，这是为什么？

其实，张无忌的愤怒是出于内心中对赵敏的爱，如果赵敏杀了殷离，自己就不能和她继续在一起了，因此恨其不争才打她耳光。敏敏对无忌的心思洞若观火，强大的洞察力给她带来无穷自信。她知道哪怕全世界都与她为敌，无忌也会出来保护她的，于是长剑出鞘，逼得无忌手足无措。

第二回合，敏敏再次在极端不利的条件下完胜！

其后两人躲入鼓中，外面斗得如火如荼。【赵敏将嘴凑到张无忌耳边，轻轻说道：“你这该死的小淫贼!”这一句话似嗔似怒，如诉如慕，说来娇媚无限，张无忌只听得心中一荡，霎时间意乱情迷，极是烦恼：“倘若她并非如此奸诈险毒，害死我的表妹，我定当一生和她长相厮守，什么什么也不顾得了。”】

一句耳边轻语，威力竟如此之大！身外则是如火如荼的争斗，两人处于极其危险的处境当中，可是，纵然如此，也没有“我要对你说我有多么爱你”来得重要！这样的情状，谁能不动情？谁能不动心？再往后，周芷若使尽各种功夫，略微稳住局势，陪着无忌到了大都。无忌来到两人初次相遇的酒店，发现赵敏居然也在，一切如故。瞬间被击倒了……

反观周芷若，此时只能一哭二闹三上吊了，失败基本无可挽回。再后来的抢亲什么的已经不重要了，因为无忌已心有所属，他需要的，只是一个冠冕堂皇的理由，比如救义父。一旦理由找到，没有了道德的羁绊，他就立刻义无反顾地投入敏敏的怀抱中。

谈三点启示。

首先，光爱还不够，要大声说出来。你不说我怎么知道你要呢？赵敏除了爱得比芷若更炽烈之外，她更加勇于且善于表达自己的爱。而且当碰到困难和挫折的时候，有人会患得患失，畏首畏尾，而赵敏选择的是更加积极地一往无前，最后她赢得爱情。

其次，光有勇气不够，还要有足够的技巧。

技巧的第一层是要懂得营造氛围，创造机会。未必要有烛光、有鲜花，君不见，赵敏同学对路边小店、海岛、游船、破庙，无不应用得得心应手。关键在于创造接触机会，让双方要么觉得十分放松，要么觉得十分刺激。无论是大排档、大马路，还是过山车、看鬼片，其实都是好环境，就看你怎么用。再比如小店的二次邂逅，多好的环境。你或许没有这样的缘分，但是完全可以侦察好地形，自己制造一场这样的邂逅。

技巧的第二层则在于敏锐的观察。每次无忌心里的想法在赵敏那里都是纤毫毕现的，然后她再从容选择以退为进、步步紧逼、单刀直入等各种战术。不过，心理观察是很难的，不是每个人都能像敏敏这样做到心理大师级的观察，但是，请冷静地用脑子去听对方说话。她说，你去死，多半不是让你去死，而是高兴又不好意思表露；她说，你很好，那你就可以真的去死了。

最后，很多时候，两个人已经势成水火，全世界都对你不抱希望了。不要顾及面子，不要担心嘲笑，不要把爱埋在心底，记住郡主说的“我偏要勉强”。这就叫真挚。

1.5

full court press

how to stuff a sleeping bag?

给 2015(让眉)

完成 2015 这四下敲击时，实实确确已是 2016 年了。开年当头的三天愁霾依旧，而我也被十余日的北京咳牵出了肺热，闭窗隐几，只沉沉病着。

检点平生这种抛付心力的活儿，似乎在病枕上是挺合宜的，几日来微信网间见了不少朋友应景的总结和计划，沉思往事时就难免也起了动念。

以日期作为岁月之刻度，很能见古人性冷之自虐。若无一个避无可避的端点，谁又能找到一个那么合理的由头去伤感所谓大江流日夜呢。

我很少用“老”这个词来形容时光流逝，因为我讨厌强加败相给任何事物——这个任何里也包括我自己。一年过去，奔三之路还有不短的行程，而

我也依旧不惮于矫情和卖萌，但站在年尾年头回顾，似乎不得不伤感地承认我或许确实和去年的此时有些不同了。

或许是因为事业的日益企稳——也或许只是长大。当初那个目无下尘的姑娘已渐渐肯于入世。

第一次有此警醒是今年夏天。当时出差去了趟云南，项目不靠谱，券商便自然亲和。忙完公务闲聊时，一个热衷于乐嘉所谓性格色彩学的小哥建议大家测一测各自的性格，我不愿拂人兴致，便如他言逐题答了。

意料之中的绿色人格，小哥听到答案却愣了一下，我问起原因，他笑着说："看李老师出项目时谈笑风生，气象不凡，跟宋美龄相去仿佛，我以为该是个女强人呢。"

一语如当头棒喝，倒把我自己唬了一跳。网络诗坛里我被惯加的称呼是"第一含羞草"，盖因每及英雄小宴时，我总是温驯讷言的；而在德国时，也很惯了被人喊着"神仙姐姐"来喻所谓不接地气——然而看来我举止的适应能力还要强于心灵的惯性，职场忽忽两年余，虽然骨子里仍厌于应酬，性子上看，倒已很能捭阖自如了。

可是，虽然"女强人"的气场令我向来羡慕，小哥这句话却依然令我不快且慌张了许久。

又后来我换了工作。

面试是处极古雅幽静的所在，分花拂柳革履翩翩，行止间就也难免有些矜持张致。待得后来跟同事们熟悉了，秘书跟我说他当时对我的印象是个"很有自信的能力型管理者"——及至这时我才无奈接受自己原来已经是一头如此成熟的大尾巴狼的事实。

此后渐渐地，我也会开始翻翻金融类各种大部头，也惯于闲暇时点进各种财经类公众号去看看各家靠谱或不靠谱的言论，更时常要跟天台、高树、杨虚白、佛爷等诗词圈的朋友一回回天南海北地从宏观大势浑侃到股债纷纭——而转过头来想想这种伪事业型人物姿态，却曾经是少年时那个矜持戒备的我多么不屑的。

我小时候对未来的想象颇得神雕侠式中二中年范儿。时常 YY 着自己戴着人皮面具在金融的江湖里飘然去来，片叶不沾身——而揭下假面后，却只见十六年来苍白清俊依旧。然而想来是因为我的江湖太过炽热，乃而方才两年光景，面具的纹路便已作用于肌理了。

许是往来见惯了香风鬓影的丽人与衣冠楚楚的才俊，对自己就也或多或少有了些不清高却矜贵的期冀吧——至今行止爱好虽犹不能俗，却也为着怕给人家说句文人多落魄，就再绝不肯一味清高地压着自己走到"非诗不可"的境地去。

好么？不好么？回头想来，虽然有点小不甘，也还须要接受了。只因比较起一个不再清高的社会人，我更反感的还是百无一用却犹自愤世嫉俗抢文学史话语权的酸丁，嗯，尚有好恶，纵然是变化着，那也很好了。

一年间，诗人的聚会虽不常有，却也零星不断。别聚匆匆，京中诸子也往往有些"相过不寂寥"的心念。点检起来，和这群人相识堪堪也已七年，

坐在筵中猛忆重头，却发现自己确然已是嗔笑自若，虽非侃侃之辈，却也绝不似当年的腼腆稚子了。八胡有次感叹说：初与众人识时大家还是聊诗的，复几年，说情场爱恨，再几年，言立业成家，而今却已是子女成行，多有不能语矣——而我作为北平圈儿里最小的一位，就也免不了要时时被他们催问起几时生子，无论最终做何答复，少时桀骜的心气却总是沉定下去了。

我倒觉得这沉定让我舒服。看着每个诗人都在所谓的市井烟火中周旋抵挡，对诗，我就也少了几分少年人固有急功近利的进取心。许是过起了步履匆匆的日子，才更加向往随机游走式的浪费与碰撞吧——这一年来，或读或写，我的自由度似乎都要比从前大得多了。

不惮于在所有自己半吊子的领域里思考和碰壁，对诗而言或许也算是一种不务正业的致敬？阅读之间不再追求于增长功力，下笔之时也不再心系名家，于是我今年写东西的时候倒是在哲学、美术和西诗里好好拧巴了一圈。蘸出来洋洋一抖脸，无论行为了什么艺术，这身段倒也近如一只高傲的沙皮狗了。

我素来不法古人，也不在意门派家数。只因我之所以愿意写诗而不仅仅是读诗，正在于自己喜欢去捕捉一些在斟酌语句时终将落于构想区间外的、衍生出来的意外灵机。这灵机虽大多时候是游离而不成形的，但也偶尔极能令我惊艳。细微的游跃仿佛历史中的一只只扇动翅膀的蝴蝶，或许将决定整首诗的走向——因为让我抓着，所以它终而存在。这种感觉的美好，没有经过思考和创作的人们恐怕是很难体会的。

于是这一年来我写诗的时候更常毫不吝惜地抛弃原始的成句去追寻一霎狂慧，然而这尝试也是有时成功，有时失败。今年我的诗写得比往年更少了，自己读起来，脑子也要拧几个弯几似要沥出水来。从文字上讲，这种写法大大影响着所谓初心的真诚，怕是入了小障的。然而这障既能让我目眩和欣喜，那就也不妨耽几年吧。

法古的诗人往往甘心顺着趋势线把文字运转下去，而不敢闻见这些可爱的方差。而读诗时，我们看到的更仅仅成了不再具有时间性的成品。拜泥塑不如心中有，这一年来我浑写浑读，隐隐觉得要写得有成就感和快乐，只怕还是应该放下一些责任心和敬畏感的。

最后，今年放下敬畏感之于我，还见于一些心结的解开。

这一年我见到了睽违多年的朋友，也在这次见面里荡没了网间交集的最后一层遗憾。倘和曾经同道的朋友经年重见时依然能相语如故，恍如渚涯岁月不曾遥隔，那么就中曾有的误解种种，便也就都成了过去式里一道释然的笑纹。

我十分很感激这种仪式感，正如我感激岁月之所以为岁月，肯于如此温柔地分离开所谓的“曾经”和“倘如”。

是的。All above。谢谢 2015。

1.6

first try spothero

1.7 things to abandon this year

several things to abandon in order to save more time this year

- weiqi video watching
- Chinese news website surfing
- fruitless chatting, dining outside

check if a ubuntu package is installed

apt-cache policy **package name**

fun sentences

继上海一男子造谣自己因造谣而被拘留 15 日而被拘留 15 日；美术馆反法西斯胜利 70 周年画展因庆祝反法西斯胜利 70 周年活动而暂停；俄罗斯一票否决了乌克兰提出的取消俄罗斯一票否决权的安理会提案之后……中国股市为预防恶化而紧急停止的熔断机制为预防恶化而紧急停止。

successfully find a proxy to run netease musicbox on Ubuntu

also learned from Luo Dan that **pptp+shadowsocks** is another approach to solve the problem

TODO I know roughly what is API. But what does the code actually look like, can you find good real world analogy?

China's broken stockmarket(from economist)

Update, January 7th, 3.50pm London: China's stock exchanges announced on Thursday evening that they would suspend use of the circuit-breakers. The securities regulator said they were not the main cause of the market's fall but had not achieved their aim and had instead caused a 'magnet effect', as described in the article below.

BIG swings in the Chinese stockmarket are par for the course. But even by its wild standards, the alacrity of its latest crash was stunning. Just 13 minutes into trading on Thursday, the CSI 300 index of blue-chip stocks fell 5%, triggering the first circuit-breaker: a 15-minute pause for traders to

supposedly regain their cool. When the action resumed, it lasted all of one minute before the second and final circuit-breaker was hit: the CSI 300 fell 7%, which necessitated a closure of the market for the rest of day. Trading, in other words, lasted all of 14 minutes before being halted.

The obvious conclusion to draw from the market sell-off is that China' s economy is in big trouble. Why else would investors be in such a rush to dump their shares? Growth is certainly slowing, but the problem with this view is that the Chinese stockmarket has only ever had a tenuous relationship with reality. It is often derided as a casino. Wu Jinglian, a veteran economist, has quipped that this is unfair to casinos. They have strict rules and gamblers cannot see each other' s cards. In China's stockmarket, the rules rarely apply to big investors, who treat price manipulation as a basic trading strategy.

But while the swings of the Chinese market defy explanations most of the time, there is actually extensive research to help explain the dynamics of the latest crash. For the culprit, look no further than the circuit-breakers that regulators introduced at the start of this week. Only four days into operation, they have already been triggered in much the same manner twice: with the 5% threshold hit first and then full closure at the 7% level soon after. The theory of circuit-breakers is that they are supposed to help calm an over-excited market. In China' s case, it appears that they have done just the opposite: encouraging traders to lock in sell orders to make sure they are the first to escape the market before the bottom falls out.

For analysts who have studied circuit-breakers, this should not be surprising. They generally fall into two camps: those who think they help to reduce volatility and those who worry they exacerbate it by leading to an acceleration of trading before halts occur. But even the former acknowledge that circuit-breakers pose the risks described by the latter. The general view is thus that that they should only be applied in extreme cases.

As Arthur Levitt, then chairman of America's Securities and Exchange Commission (SEC), explained in 1998:

Circuit-breakers were meant, from their inception, to be triggered only in truly extraordinary circumstances—ie, a severe market decline when the prices have dropped so dramatically that liquidity and credit dry up, and when prices threaten to cascade in a panic-driven spiral. As long as the markets are closed or have the potential to close early, there is uncertainty. Uncertainty for individual investors leads to confusion. In China, big swings between the open and close of the stockmarket used to be the norm, much to the chagrin of reporters who were expected to divine something intelligible from the movements. Peter Thal Larsen of Reuters Breakingviews put it

best in a tweet: “Iron rule of Chinese stockmarkets: any observation based on intraday movements will be obsolete by the close of trading.”

The introduction of circuit-breakers has changed this logic. They have highlighted a problem known as the “magnet effect”. The Hong Kong Securities and Futures Commission defined this in a 2001 article as the possibility that circuit-breakers might “accelerate price movements towards the preannounced limits as market participants alter their strategies and trade in anticipation of a market halt”.

Based on the four days in which China’s breakers have been in operation, the first magnetic pull seems to kick in at around 4% down. Traders rush to sell before they are locked out. After trading starts again at 5% down, the magnetic draw to 7% is almost irresistible; no one wants to be left holding the hot potato.

In its design of the circuit-breakers, China has violated one of the basic principles of those countries that also apply them: the gaps between breaker levels should be sufficiently wide to avoid having an overwhelming magnet effect. The SEC halts trading at the 7%, 13% and 20% thresholds for the S&P 500 index. And bear in mind that American markets are far more mature, making even 7% changes a rarity. In China, the 5% threshold is something that was crossed with regularity before the circuit-breakers were introduced, with the market often giving up its gains or paring its losses over the course of the frenetic trading day. Now, though, the circuit-breaker makes those lurches permanent, until the next trading day begins.

None of this means that the Chinese stockmarket should be performing well. Share prices, especially for small-cap stocks, are still extremely frothy. But the madness of 14-minute-long trading days was utterly avoidable. The latest update is that the securities regulator has called an unscheduled meeting to discuss the circuit-breakers, according to Bloomberg. If only they had bothered to discuss them properly before implementation.

arrive in Chicago, join with Wenbin

1.8 Reunion with Dongming at Purdue!

1.9 revisit Chicago in rain & snow

Willis tower(didn’t climb), Millenium Park/Cloud Gate, Art Institute, 名轩（粤菜）

until you, song

1.10 visit Jinwei at Notre Dame in snow, revisit Chicago in better weather, reunion with Xuecheng, Jun & Yitong

1.16

练刀工

作者：灰子来源：知乎

1. 刀具和案板做中餐，基本上 3 把刀就够了。砍刀是多大块的骨头，火腿啊、排骨啊之类的。剁刀一般用来剁鸡、鸭、兔子之类骨头小的禽类，还有就是剁肉馅、剁姜蒜的时候用。切刀用的最多，素菜和没有骨头的肉一般都用切刀。案板要大一点的好，因为大案板更稳，在厨房里一般不用案板，用墩子，二十多厘米高，就像是一树墩子，多形象。如果觉得自家的案板不稳，总是晃动，可以在案板下面垫一块帕子，就稳多了。2. 磨刀磨刀石有两种，一种是砂石，一种是油石，砂石粗糙，油石更光滑细腻。刀太钝了，就先用砂石来粗磨，刀刃的角度磨好了，再用油石来磨地平滑。刀与磨刀石夹角为 30 度左右，刀刃太薄了很脆，太厚了不够锋利。我之前在餐厅切菜，每人有自己的专用刀，我们几乎每天都要磨刀，最多 3 天要磨一次，不然自己的菜刀太钝了，要被同事们鄙视。但在家里不必这么频繁，毕竟我们当时一天要切七八个小时的菜。磨得特别棒的刀，拿一个稍微软一点的土豆，削去皮放在案板上，把刀立上去，手握着刀把，几乎不需要用力，刀靠着自身的重力就能切下去一半。（这也是因为餐厅里的刀本身就比家用刀更大更重）3. 刀法刀法有很多种，入门的刀法，会几种就够了。直切，垂直下刀，干净利落，绝大多数蔬菜都都是直切法。推切，比较柔软的食物，比如绝大多数肉类，直着下刀，肉的形状要走样，边向前推边下刀。推拉切，特别硬的东西，比如冻牛肉、大头菜之类的，还有就是特别易碎的食物，比如面包、馒头，要用推拉切，方法类似于锯。花刀，餐厅里做的鱿鱼卷、松鼠鱼、肝腰合炒之类的，原料要上花刀，一般来说，每一刀要切到原料的 3/4 处，但一定不能切断，属于难度比较大的刀法。剁，肉馅啊、姜蒜末啊。我们在厨房里剁姜蒜，有时候一次十几斤，两把刀左右手一起剁，剁熟练了能剁出马奔跑时马蹄发出的“哒——哒哒——哒——哒哒”的声音，节奏特别动感过瘾。砍，主要对付大骨头，刀要举过头顶，猛地砍下去，砍下去的速度要快，还需要花些力气。三刀都能砍到同一个位置，你就出师了。其实，也不必我讲刀法的理论，实践中大家自然会找到最适合的刀法。4. 切菜的姿势两脚自然分开，上半身微微前倾，但不要弯腰，案板的高度大概在腰部，身体至少离案板一拳。右手握刀，大拇指自然弯曲，左手放在食物上，指头弯曲，用中指第一个关节顶在最前面。刀垂直下刀，抬起刀的高度永远不要超过左手中指的高度。如果你能一直保持这样的姿势，从理论上来说，你就永远也不会切到手了。5. 终于可以开始切菜了以切土豆丝为例吧。左手按稳土豆（把

土豆比较平的一面放在案板上），右手垂直下刀，切下了的土豆片，上下要一样厚薄。注意要使用腕力下刀，不要用胳膊的力气。切一刀后，左手中指第一个关节往左退 2 毫米（别真拿尺子去量哈，凭感觉），再重新下刀，眼睛看着刀的右边。关节连着退 3-4 次后，把整个左手向左退一小截。土豆切掉 1/3 左右，把切面平放在案板上，这样土豆就特别稳了，继续切。土豆都切成片以后平铺码好，按照同样的方法切成丝。（写着写着，就怀念起当年一帮同事一切比赛切土豆丝的时候，谁都不服谁，我当时的技术还是不错的，平均 2 分钟可以切出一盘比火柴棍差不了太多的土豆丝，现在手生了，不行了。）再说说切肉。肉难切的原因在于比较柔软，容易跑偏。切肉的时候，左手要按得稳一点，下刀要边向前推便下刀，利索果断一点，切下去后刀往右边偏一下，切好的那片肉就掉下去了。冻得不那么硬的肉，要容易切得多。一块肉切了 90%，最后一小坨，是最难切的了，这里要引入一个名词，叫片，可以把肉平方过来，刀也与案板平行，这样推拉着切，叫片，技术要难一些。6. 怎样把刀工练好熟能生巧，刀工是一项技能，智商再高，技巧再好，不练出手感来，刀工是好不了的。认真切，切菜是一件很无聊的事情，很容易跑神，边切菜，边聊个天想个心事啥的，当然，这样除了容易切到手以外，也没啥坏处。但如果想切得好，同样要认真才行，切菜的时候把注意力集中在两只手上，很快就能找到手感。

order

作者：谢熊猫君来源：知乎

中餐炒菜的大概炒菜顺序：1. 烧热油和锅 2. 大葱、生姜、蒜、各种椒，爆香 3. 放主料炒脱生 4. 料酒、酱油、醋，去腥上色调味 5. 家常菜可以加高汤或者水焖熟 6. 盐、糖、味精、鸡精等调味 7. 勾芡 8. 撒葱花以上步骤和料不是每道菜都必须要，看你炒什么菜调整用料、步骤、顺序。

请不要追问油多热算热、盐怎么放、各种调料放多少这类问题。参考《做饭要快，无他，唯手熟尔。》

1.17

给丫的信

本来有些话是想当面对丫说的，不过一来电话几次想约你打不通，二来觉得发普通信息也不够庄重，所以选择了信的方式。

我是一个不喜欢矫情、直来直去的人，所以这话也大大咧咧的说————在波士顿相处的那几天里，丫让我心动了。我喜欢丫。这份喜欢和你姐他们无关，虽然如果没有他们的暗示和鼓励，我未必会真的会想到表白。因为心动，所以去看过你之前的微信还有校内能看到的状态，想多了解丫。看过后，我觉得，丫是个很有性格的姑娘，虽然在杨叔叔那不是那么明显：）

“自古表白多白表，从来姻缘少原因”。川哥毕竟也二十八岁了，知道这些道理。从你的回应，我知道你对我是没什么感觉心有回避的。可也有句话，叫做“说了后悔总比不说后悔来得好”。既然是自己真实产生了的心意，那么去做忠实地自我表达就不是件坏事。

我知道我大你五岁，可咱这都什么时代了，五岁无非也就是半辈人，而且重要的是我觉得我的心怀依然少年。我知道你决定了回国，可那天车上我也和你说过，文化上我对美国没什么留恋的。除了父母，我几个最好的朋友都在国内，我看的清留美和回国的得失。如果是和喜欢的姑娘一起探索未来，我愿意。我知道我学理你学文，可我觉得这其实可以互补，况且我这个人骨子里其实感性，在理解别人这点上悟性不差。当然，如果我是长相上没过你心里的阀值，那就该一票否决了。作为外貌协会会员，我完全理解 ~

我相信，这世界上真正美好的东西，都应该是简单明快的，对便对了，不对便是不对。人与人间的理解有天然的局限，可若不去做真实明快的表达，是永远也无法突破那层局限，而实现心与心间相对理解的。你对你自己的心有最好的判断，我这封信本质上是为了我自己的忠实表达，你读的时候可不需要烦恼啥，若是能会心一笑那是最好了！

总之，我希望你我之间的表达是单纯明快的。如果你对我没有感觉，那就直说，我这人明白“一个巴掌拍不响”，看得清这缘分该怎么惜, 从此以后，做你合格的朋友、大哥就够了。如果你觉得我们有希望, 但还需一些时间方能开始，那就我追你。

好了，写到这我想我试图表达的也大致清楚了。于我来说，循心而往，随缘而去，纵是难成美眷也不负少年心怀。虽然未必能打动你，可你对我可能有的误会也该消解了。和丫这样的好姑娘认识，是我有缘。无论如何，做你合格的一位朋友和大哥，是我可以完成的承诺 ~

最后，祝丫在 LA 玩的开心。等你回来，欢迎你来我的地盘，我一定带你好吃好喝好玩：）

川哥二零一六年一月十七日夜于新泽西

1.20

1st 回信

川哥：

谢谢你写这封邮件给我，抱歉我看到晚了。

我一直都处于一个并不想谈恋爱的状态，这么说也不是在敷衍你，因为我从来都没有谈过正式的男朋友。你从我姐姐她们的态度就可以看出来，她们都很希望我可以和一个很好的人开始一段感情，大概是因为她们怕我拖着拖着就剩下了把。但我现在依然没有这方面的想法，现在的状态我就很舒服了。

我其实是一个非常慢热的人，要和别人成为朋友前需要很长的时间。这

回在波士顿认识你们当然是很愉快的，但是对我而言其实是一个很不自然的状态。就像你说的一样，我在我舅舅家并不是真正的性格，这样的相遇真的是很遗憾。我想，如果我们是慢慢熟悉起来，在了解对方的喜恶之后再这样长时间的相处大概会更好一些吧。就像“丫”这个称呼，我其实一直不喜欢这个名字，也就是家人会这样叫我，我的朋友同学没有一个人是知道的。每次家人以外的人发现这个称呼都让我觉得很难堪。我也应该给你说声对不起，在你访问了我人人网之后把能屏蔽的都屏蔽了实在是不礼貌的做法。我不是对你有什么意见，我只是一直都不希望别人了解到我还没有准备好让他们了解的地方。人人网的账号我一直都是想删除的，只不过忘了注册邮箱的密码我删不掉。我以为你发现以后应该会讨厌我，不理我了，没想到你还是对我这么耐心，谢谢你啦。

我从来没觉得你大我五岁或者文科理科会是个问题。只是和你的热情相比，我实在是一个太过冷漠的人了。我习惯和大部分人相处都保持着一段距离，整个家里的平辈，我真正关心的就只有这一个姐姐。我也讨厌维持异地的感情，无论是友情还是亲情。哪怕是和旦旦姐姐，我们不在一个城市的时候也基本上是不联系的状态。所以我如果先你一步回国，我实在是不敢保证什么。退一步说，我们两个根本不算是真正的了解对方，你要放弃在这边这么多年的积累贸然和我回国，我良心上都会过不去。

你骨子里是一个很感性的人，但我其实很理性。我没有觉得你不好，我就是觉得完全不可行。其实要说谈段恋爱也不是不行，毕竟我家人都有点着急了。可这样做的话就太自私了，毕竟我觉得不可行那我就不会投入，而你在现在的这个年龄，应该是会想谈认真严肃的感情了把。我和我姐姐她们一直都觉得你是很体贴很真诚的人，我也不希望到了最后结下一个仇人。

你看，真正的我和你想的大概完全不一样吧。要是我们是很自然的相识相知，这些麻烦尴尬可能就都不会有了。我其实也不习惯对着不熟悉的人做这样的自我剖白，把这么讨厌的自己直接展示给别人看。毕竟如果只是做朋友，很多事情根本就不需要了解到的。不过要是你觉得哪怕我这么讨厌都还可以做朋友的话，以后就叫我一同或者直接说“你”就可以了。真的很谢谢你。

art of the command line

some examples:

1. bash 下，用 tab 补全，ctrl-r 搜索命令历史
2. ctrl-w 删除当前命令的一个单词，ctrl-u 删除到行首，ctrl-k 删除行尾，ctrl-a 移动到行首，ctrl-e 移动到行尾
3. ctrl-x ctrl-e 可以调用自己定义的编辑器来编辑当前命令行，当你要写一串超长的命令的时候就知道有多好用了

4. cd -回到上次的目录
5. 输入命令不想立即执行，但是又想保存到 history 中供搜索，怎么办？ctrl-a 跳到行首，加一个 #，回车，搞定
6. pgrep 搜索进程
7. nohup 让进程忽略 HUP 信号，disown 让后台启动的进程忽略 HUP
8. 自定义命令别名，alias ll='ls -latr'
9. 用 tmux 或者 screen 来让你的进程不会因为 ssh 连接终端而被干掉

2nd letter

一同（这么称呼现在有点别扭，我慢慢改）：

首先，我想说，一个人选择单身可能有很多原因：觉得一个人状态不错，没有合适的人，或者不在合适的时间，等等。但不管是哪种，其他任何人都没有权利去强迫她/他开始。就是我的老同学里都还有几位非常独立的女性，到现在都还享受着单身的状态（问过她们基本说目前的目标是多赚钱 = =），我特别欣赏。而你才二十三呢，完全没有必要因为家庭和周围的压力去勉强开始一段恋情。至于我个人，虽然家里在催了，不过也就是在老妈面前说好好好哄哄她，骨子里我不会急，觉得急了更没用，要找一定还是要找自己真的喜欢的，不能随便就让他们给介绍了。找不到的话，一个人也有一个人的过法。

总之，在我看来，恋爱/结婚与否，何时开始，和谁开始，这些归根结底应该是一个人的独立选择。虽然因为历史、文化、地域的限制不总是这样，但一个人发自内心地愿意，主动地做出的选择，在大部分时候都要比被迫去选择的情况结果要好的多。

你说“这样的相遇真的是很遗憾。我想，如果我们是慢慢熟悉起来，在了解对方的喜恶之后再这样长时间的相处大概会更好一些吧”，前半句在我看来并不是这样。我想，我们这样的相遇是奇妙的。其实这个圣诞我本来是想回绝杨叔叔的，而我也确实原定二十三号开车去普度。可不曾想到我那边的朋友临时和我说他要一月七号才从国内回来，我只好把行程改到那天。然后二十号左右杨叔叔又给我打了几个电话，挨不过他的热情，最后一次我问他有没有同龄人（当时想不然我一个小辈过去要是面对的都是他这样的长辈咋办），他说有，然后把你联系方式告诉我看我能不能和你一起过来。所以才有了之后的相遇，在我看来还真有些奇妙。当然也不是相遇那天就产生了追你的想法，那天只是觉得你是个可爱的小妹子（帮女生拎箱子啥的是我向来的作风＝＝），真正有了这个想法是在你们送我礼物的那天晚上（当然后来我也猜到了估计多半是你姐的主意）。

人和人的相遇很难预先设定一种方式，更别说是最好的那种了。我们既然这样相遇，那就是缘分。不管命运如何流变，我会去把握我所能把握的，我想，到回首的时候，我们一定已经成为了相知相惜的朋友。所以啊，哪有什么相遇的遗憾，我们的相知才刚刚开始呢。

你说你”太过冷漠“，首先我并不觉得，其次能消融冷漠的不也就只有热情吗？难道要以冷漠对冷漠？不管是恋人的相处还是朋友相处，出现了问题，总有一个人要主动先走出一步才能解决问题。至于”慢热“，可解者唯有时间和相伴。

你还提到了恋爱的可行性。我理解你觉得不可行的最主要原因一是还不真正的了解对方，二是地点——你决定了回国，而我目前在美国好好的，两人很难在一个地方开始。我是这么想的：

我本来就想和你相知，所以去真正的了解对方本就是要做的。所以这个原因并非不可行的绝对理由，我们边接触边了解对方，如果到了某个点我们有了默契，这个自然就不是问题了。至于地点，这个有可能成为不可行的绝对理由，因为两个人最后若不能长时间在一起，谈什么恋爱嘛！所以在我的理解，可行与否关键在于地点。不知你是否同意。

其实我有回国的想法很久了，自从两三年前我想清楚毕业后不做学术去业界之后，毕业后长期留美对我的意义就不大了，因为文化上我真的对美国没啥留恋的，也就只有做学术让我觉得非在美国不可。在遇到你之前我的想法就是毕业后如果这边有好的 offer 那就工作一两年回去，不会拖到三五年；如果没有好的 offer 那就直接回国，理想的城市是珠三角那三个以及上海（北京的雾霾让我没那么想去），珠三角是因为离家近同学朋友多，上海是因为如果去那边做金融的话有不少好的 connection 能用上。当然我也想到了唯一的变数——未来可能的那位姑娘，如果她是想留美的，那我想我也会决定留美；当然如果她想回国，那自然就按之前回国的想法走。我不知道你回国后最想去哪，现在你也不必告诉我，因为我们还不熟：）我和你说我的想法是想告诉你，如果我们在一起然后我回国了，也并没有为你特别放弃什么牺牲什么，你完全没有需要心里过不去的地方。

说到这我想可以理一理了。你觉得现在的状态就很舒服，我其实也觉得我自己现在一个人的状态不错（虽然能和喜欢的姑娘开始是更让人开心的）。目前至少我们可以在做朋友的过程中去把可行性中的第一个问题解决掉。最后能不能两个都解决，使你觉得可行，那我就尽人事而安天命吧。

既然试图去解决，那我自然就要提议咯。我不知道你现在想何时回国，我就假设在四五月吧。那这样的话在你回国前大概还有三个月，三个月中我觉得你应该抓紧时间再去多看看美帝的大好河山。我现在上班是 part time，所以每两周出去走走都是有时间的。在美东这边自然风光的话尼亚加拉大瀑布，仙那度 (DC 附近)、大烟山（田纳西州）国家公园都很不错，大瀑布之前听你是想去的，而实际上这三处我都没有去过。我想在三个月中，我推荐你可以考虑二月或者三月去尼亚加拉大瀑布，四月初樱花节的时候去 DC 看看樱花然后顺道去仙那度。至于大烟山，五六月份的时候那儿有共鸣萤火虫

的 festival（而且从我们新泽西这去那的路上可以开美国最漂亮的国家景观公路——蓝岭高速），可如果你回国的话我不知道你来不来得及了＝＝　你可以约上你的一个同学或者朋友（如果住店正好你们 share），然后我们三或四个人（我也可以再叫上一个我的朋友和我 share 房间）一起 road trip 这几个地方。至于小一点的地方，我之前也和你提了普林斯顿值得一看，这个地儿等你从 LA 回来后咱随便找一周周四到周末的时间就能带你当天游遍。

远一点美西的话我去过的地方不少（我主要是和基友刷国家公园去的），如果你有想去但还没去的也可以告诉我，我刚说了我现在 part time，每两周都有一周的时间可以自己支配，只要合理的计划，去西部也是没问题的。

平时的话如果周末你有空我也会找时间约你吃饭或者去看看其他的（比如百老汇看剧）。

恩，能想到的暂时就这么多。相信我现在的想法通过这信都能传达到了。

2nd 回信

不好意思川哥，可能我昨天没有说清楚。我现在真的一点都不想谈恋爱，想交朋友也是想交只会成为朋友的那种。还有我二月底就会回国，所以时间上可能真的来不及了。不好意思

last response

明白啦。等你 LA 回来还是告诉我，说好带你去普林的 ~~

1.21

“Rebounding isn’t about the size of your body, it’s about the size of your heart。”

body strength training

- 引体向上:pull-up very efficient way to improve strength!
- rope skipping

Plan to prepare code interview

- read **The Algorithm Design Manual**
- read **Cracking the code interview**
- **TODO** Princeton 2016 spring Algorithm Course

reconcile with Leila

1.23 Ex Machina

python code in movie Ex Machina

```
#BlueBook code decryption
import sys
def sieve(n):
    x = [1] * n
    x[1] = 0
    for i in range(2,n/2):
            j = 2 * i
            while j < n:
                    x[j]=0
                    j = j+i
    return x

def prime(n,x):
    i = 1
    j = 1
    while j <= n:
            if x[i] == 1:
                    j = j + 1
            i = i + 1
    return i - 1
x=sieve(10000)
code = [1206,301,384,5]
key =[1,1,2,2,]

sys.stdout.write("".join(chr(i) for i in [73,83,66,78,32,61,32]))
for i in range (0,4):
    sys.stdout.write(str(prime(code[i],x)-key[i]))

print
```

1.24 编程入门指南

编程入门指南 v1.4 by 萧井陌

编程入门指南 v1.4

阅读此文最基本的收获即是————编程能力的提高是三位一体密不可分的：计算机硬件系统原理，算法和数据结构以及编程语言。文中开出了不少很好的书目和资源，我会尽量选择性地拿来精读或略读。

1.25

超市结账请留心打折商品

今天在 Hmart 买笋，看到原价 5.99/lb 而现在特价 3.99/lb 所以买了两个，check out 后回家看 receipt 发现实际还是按 5.99 收的。以后对这种特价或者打折的东西在最后结账时一定要仔细确认一遍。我之前在 Hmart 也遇到过这样的事，看来 Hmart 可以放入黑名单了, 除非在其他中国超市买不到的或者没有更新鲜，以后尽量不去那 ~~

Mountain Blanc in Alps

Know about it from today's google frontpage recommendation

got response from Yitong

1.26 The Witness

the witness game by Jonathan Blow

- some players' remark
  - 1 Here' s a Hint, **Think outside of the box**. Not only think but walk around the box, Think what the box is? Think how did the box get here? What does the box think about me? Do you love the box? Does the box love you? Then walk away from the box because you have no idea what you' re doing. - Luke Rising 2016
  - 2 This game is not going to be for everyone. If you're looking for a game that has action, violence, and an engaging plot that will keep you on the edge of your seat, look elsewhere. If you're impatient, and if you don't want your brain to hurt, this isn't the game for you. With that being said, in the amount of time I've invested in The Witness so far, it's already one of my favorite puzzle games of all time. The game takes the basic concept of maze puzzles, and introduces new mechanics to them to make you think. It then takes those new mechanics, and gives them a little twist, to make you think even more. And then once you

think you've mastered that mechanic, it mixes it with another mechanic that you've previously learned, or even a brand new one. This game is hard, and it doesn't make any attempt to hold your hand. This game is going to make you want to smash your head onto your desk, but in a good way, because you want to conquer this latest puzzle so badly.

The game has an incredible atmosphere, with gorgeous visuals and ambient sounds immersing you into this island world. There aren't any graphics settings, which is a huge bummer, but hopefully that is something that could be patched into the game. The vibrant colors of the island really pop off the screen, and the different locations throughout the game are really neat. You can also listen to some audio diaries located throughout the island, which have different quotes from all sorts of different people. This would be a great game to have a podcast on in the background. Not that the sounds are bad, they do a good enough job to keep you immersed in the game while you're playing, but it might be nice to have something else to listen to as well. The controls work great, on both a controller and a keyboard and mouse. Pick whichever one you're more comfortable with. I know a lot of people have been complaining about the $40 price tag, but from what I've played so far, it seems absolutely worth it. There are a ton of puzzles in here, and it's going to take you awhile to beat the game.

tl;dr I you want something different than a complex puzzle game that keeps you on your toes with a great atmosphere, you're going to want to look elsewhere. But if you've been craving a game like The Talos Principle or Portal, you should absolutely pick this up. This game is worth the price tag, and it's going to be a long time before you forget your time in The Witness.

HTML basics

<!DOCTYPE html>   <div id= class=> <h1> </h1>

- to specify the related CSS file <link rel="stylesheet" type="text/css" href="../style.css" />

CSS basics

Does that interest you?

How to deblur an image?

前面的回答都是关于抓包，如何截取还没有经过模糊的图。我打算讲一个更通用的，即便服务端给了模糊的图，照样能还原。不要以为 CSI 里的照片还原技术只是个特效。目前所用的模糊，基本都是卷积，而且几乎都是用 gaussian kernel 卷积。而根据卷积定理，离散信号 x 与 y 的圆周卷积对偶于频域上 x 与 y 离散傅里叶变换（DFT）的乘积。用公式表达就是 …… 所以，gaussian blur 可以对原图和 gaussian kernel 的图像做 DFT，按像素乘起来，再做一次逆 DFT。这种方法和 gaussian kernel 的半径无关。所以一般在需要超大半径模糊的时候，速度快于传统的 separatable gaussian blur。用 DFT 做卷积的另一个好处是，同样用这个算法/代码，就能做反卷积！改成 DFT{x} / DFT{y}，其中 x 是已经经过模糊的图像，y 仍是 gaussian kernel 的图像，照样那么做，就能得到清晰地图像。唯一需要的参数就是 gaussian kernel 的半径。这个需要根据实际情况动态调整。做好的话，就能像 CSI 那样做实时连续的模糊到清晰的还原。当然，因为在模糊的过程中很多信息已经丢失，这里得到的是个可能的原图。换句话说，存在两个不同的输入，模糊后图像相同。这时候用这个算法是无法得到正确结果的。而且，FFT 可以在 GPU 上实现，即便是手机级别的 GPU 也能做到接近实时。所以，可以做一个 deblur 的 camera app，然后，大家都知道怎么用了。

1.27 Useful inequalities in statistics

Useful inequalities in statistics

Here then, cerebrating, is a list of inequalities I would wish to know, if I were a graduate student working on statistical theory today. They are generally grouped by topics; analysis, matrices, probability, moments, limit theorems, statistics.

1 Cauchy–Schwarz 2 Jensen 3 Hölder and triangular 4 Fatou 5 Bessel 6 Hausdorff–Young 7 Basic Sobolev inequality in three dimensions only 8 Frobenius 9 Sylvestre 10 Determinant bounds, e.g., Hadamard 11 Kantorovich 12 Courant–Fischer 13 Boole’ s inequality, from both directions 14 Chebyshev and Markov 15 Bernstein 16 Hoeffding in the Rademacher case, 1963 17 Bounds on Mills ratio from both directions 18 Upper tail of Binomial and Poisson 19 Slepian’ s lemma, 1962 20 Anderson’ s inequality on probabilities of symmetric convex sets, 1955 21 Rosenthal, 1970 22 Kolmogorov’s basic maximal inequality 23 Basic Berry-Esseen in one dimension 24 Le Cam’ s bound on Poisson approximations (Le Cam, 1960) 25 DKW

with a mention of Massart' s constant (Massart, 1990) 26 Bounds on expectation of normal maximum from both directions 27 Comparison lemma on multinormal CDFs (Leadbetter, Lindgren, and Rootzén, 1983) 28 Talagrand (as in 1995, Springer) 29 Inequality between Hellinger and Kullback–Leibler distance 30 Cramér-Rao 31 Rao–Blackwell (which is an inequality) 32 Wald' s SPRT inequalities.

Truly going back to my student days, I recall how useful matrix inequalities were in that period, when linear inference was such an elephant in the room. Inequalities on CLTs and metrics played pivotal roles in the sixties, and then again, as the bootstrap and later, MCMC, emerged. Concentration inequalities came to the forefront with the advent of empirical process theory, and then as high dimensional problems became important. It seems as though the potential of analytic inequalities in solving statistical and probabilistic problems hasn' t yet been efficiently tapped. The recent book by Peter Bühlmann and Sara van de Geer (2011) has many modern powerful inequalities. There are of course new editions of the classics, e.g., Hardy, Littlewood and Pólya (1988), Marshall, Olkin and Arnold (2011).

Quite possibly, on another day I would include some other phenomenal inequalities, and drop some that I chose today. Can anyone vouch that Efron–Stein (1981), Gauss (for unimodal distributions), FKG (Fortuin, Kasteleyn, Ginibre, 1971), Chernoff ' s variance inequality (1981), or a basic prophet or log-Sobolev inequality, or even a basic Poincaré, need not be in the essential list? Defining what is the most useful or the most beautiful is about the most hopeless task one can have. Beauty and use are such indubitably personal choices. We have, in front of us, an ocean of remarkable inequalities. You can' t cross the sea, said Nobel Laureate Poet Tagore, merely by standing and staring at the water. I figure I need to jump!

Looking forward to see the match of Lee Sedol vs AlphaGo this March!

Feb

2.4 Princeton vist with Yitong & the movie Room

2.5 学而不思则罔，思而不学则殆

thoughts on reading books like 数学与联想，啊哈灵机一动

近来想起了当年读《数学与联想》，《啊哈灵机一动》,/Basic Algebra/(Jacobson) 这些书时探索心超强求知欲旺盛的时候。也有点重新发现了如何读书如何求

知。加上最近读 Rubin 的 *Causal Inference* 以及 Lugosi 的 *Concentration Inequlities* 很有感觉，使我相信自己仍能做一个热爱探索求知的人。

学, 思还有 表达三者的关系

火花卷首语

塞林格写过一部名作叫《麦田里的守望者》，里面的主人公是一个被学校开除的中学生，他貌似玩世不恭，厌倦现存的平庸的一切，但他并非没有理想。他想象悬崖边有一块大麦田，一大群孩子在麦田里玩，而他的理想就是站在麦田边作一个守望者，专门捕捉朝悬崖边上乱跑的孩子，防止他们掉下悬崖。

我很喜欢“守望者”这个名称，它使我想起守林人。守林人的心境总是非常宁静的，他长年与树木、松鼠、啄木鸟这样一些最单纯的生命为伴，他自己的生命也变得单纯了。他的全部生活就是守护森林，瞭望云天，这守望的生涯使他心明眼亮，不染尘嚣。“守望者”的名称还使我想起守灯塔人。在奔流的江河中，守灯塔人日夜守护灯塔，瞭望潮汛，保护着船只的安全航行。当然，与都市人相比，守林人的生活未免冷清。与弄潮儿相比，守灯塔人的工作未免平凡。可是，你决不能说他们是人类中可有可无的一员。如果没有这些守望者的默默守望，森林消失，地球化为沙漠，都市人到哪里去寻欢作乐，灯塔熄灭，航道变为墓穴，弄潮儿如何还能大出风头？

人做事情，或出于利益，或出于性情。凡出于性情做的事情，大都仅仅是为了满足心灵需求，如写作、艺术欣赏、交友等等。我们做这个主页也是如此，愉快是最基本的标准。如果什么时候我们在这里找不到愉快，就必须怀疑是否有利益的强制在其中起着作用，使性情生活蜕变成了功利行为。火花是这样一个地方，它不想在踌躇满志的游戏精英网站中挤自己的一块地盘，而是很安静的，与世无争的，但也因此在普遍的热闹和竞争中有了存在的价值。我们只想开辟一块园地，让火炎玩家们找到回家的感觉。

如果一个人有自己的心灵追求，又在世界上闯荡了一番，有了相当的人生阅历，那么，他就会逐渐意识到自己在这个世界上的位置。世界无限广阔，诱惑永无止境，然而，属于每一个人的现实可能性终究是有限的。你不妨对一切可能性保持着开放的心态，因为那是人生魅力的源泉，但同时你也要早一些在世界之海上抛下自己的锚，找到最合适自己的领域。一个人不论伟大还是平凡，只要他顺应自己的天性，找到了自己真正喜欢做的事，并且一心把自己喜欢做的事做到尽善尽美，他在这个世界上就有了牢不可破的家园。于是，他不但会有足够的勇气去承受外界的压力，并且会有足够的清醒来面对形形色色的机会的诱惑。我们当然没有理由怀疑，这样的一个人必能获得生活的充实和心灵的宁静。

need to install *ispell*

你喜欢哪种类型的数学和统计？

2.6 饺子的包法

饺子的包法

2.7 过年

新年对自己的承诺

- 合格的完成 thesis 毕业！
- 在自己真正想做的领域找到一份好的工作，不必是大牛公司，但希望是有好同事能很快成长学习的地方
- 成为一个更好的 programmer!
- 成为更好的 problem solver!
- 感情有进展

狼人杀的策略

2.8 The Danger of Copy-and-Paste Learning

Successfully install and setup the ispell problem(Aspell) on my Windows 10

Note that on Ubuntu, the **ispell** is installed by default.

- From EmacsWiki Once you' ve set up GNU EmacsW32 you will probably want to add spell checking ability. aspell is not an especially good choice; but alternatives at this point are unclear. True, it integrates seamlessly with Emacs; but the last version of it for Windows was compiled in 2002 and is hopelessly out of date.

  Although it might be out of date it seems to work well. I installed Aspell according to the instructions below, yesterday (today is 2015-09-23), and it works fine, in a quite recent Emacs 25. – MaDa

  Setup for 64-bit Windows 7

  Next we need to make a series of changes in your InitFile. You need to add the path of the aspell exec to your emacs exec-path. I tried the path string without the C: at the beginning but it did not work consistently.

(add-to-list 'exec-path "") We need tell emacs to use aspell, and where your custom dictionary is.

(setq ispell-program-name "aspell") (setq ispell-personal-dictionary "C:/path/to/your/.ispell") Then, we need to turn it on.

(require 'ispell) Lastly you need some way of invoking it. "M-"$isthedefaultmethod, whichwillche$ will check all words within the region. However, I like to customize all the keybindings. So, here' s an example to use it with FlySpell:

(global-set-key (kbd "<f8>") 'ispell-word) (global-set-key (kbd "C-<f8>") 'flyspell-mode)

The spacemacs manual inside spacemacs is a good resource to learn org-mode!

check it out inside spacemacs yourself!

install clojure(inside lein) on Ubuntu

to call clojure in CMD, enter **lein repl**

spacemacs shell layer

SPC 'Open, close or go to the default shell SPC a s e Open, close or go to an eshell SPC a s i Open, close or go to a shell SPC a s m Open, close or go to a multi-term SPC a s t Open, close or go to a ansi-term SPC a s T Open, close or go to a term SPC m H browse history with helm (works in eshell and shell) C-j next item in history C-k previous item in history

罗平县

罗平县是中国云南省曲靖市下属的一个县，位于滇、黔、桂三省交界处，素有“鸡鸣三省”、“滇黔锁钥”、“滇东明珠”之称。

setup the auctex & Okular syncing feature on Ubuntu spacemacs

- the script ;; set up tex pdf viewer (cond ((string-equal system-type "darwin") (progn (setq TEX-view-program-selection '((output-pdf "Skim"))))) ((string-equal system-type "gnu/linux") (progn (setq TEX-view-program-selection '((output-pdf "Okular")))))

;; sync between auctex & Okular (setq TEX-source-correlate-mode t) (setq TEX-source-correlate-start-server t) (setq TEX-source-correlate-method 'synctex) (setq TEX-view-program-list '(("okular –unique %o#src:%n`pwd`/./%b") ;; do you understand this line? ("Skim" "displayline -b -g %n %o %b")))

The Danger of Copy-and-Paste Learning

I have mentioned above that for small code, you can copy-and-paste and tweak to learn what the code does. However, make sure that you understand what the code does, line by line. This might not help you now but it will in the future. At this point, try to explore what the default settings of the LaTeX classes book, article and report have to offer. In most cases, you will be working with the article class a lot.

What helps me when learning a new code is I comment on what each line does. I find this helpful especially coming from a non-programming background. There are instances, too, that copy-and-paste approach introduces invisible characters that introduce errors. In the long run, typing the code character-by-character, line-by-line makes you think more about what goes into your code and what fix you can do in instances of errors.

Revisit Emacs' bookmark feature later

Emacs bookmarking makes use of three things that are related but different: a bookmark list, a bookmark file, and a bookmark-list display. Understanding these is important to using Emacs bookmarks. They are explained at Bookmark Basics.

Some bookmarking commands to get you started:

'C-x r m' – set a bookmark at the current location (e.g. in a file)

'C-x r b' – jump to a bookmark

'C-x r l' – list your bookmarks

'M-x bookmark-delete' – delete a bookmark by name

Your personal bookmark file is defined by option 'bookmark-default-file', which defaults to '~/.emacs.d/bookmarks' in the most recent Emacs versions and to '~/.emacs.bmk' in older versions. The file is maintained automatically by Emacs as you create, change, and delete bookmarks.

The bookmark list (buffer '*Bookmark List*') is like Dired or Buffer-Menu for bookmarks. You access it using 'C-x r l'. (Emacs sometimes calls it the "bookmark menu list", which is a misnomer.)

Some keys in '*Bookmark List*':

‘a’ – show annotation for the current bookmark ‘A’ – show all annotations for your bookmarks ‘d’ – mark various entries for deletion (‘x’ – to delete them) ‘e’ – edit the annotation for the current bookmark ‘m’ – mark various entries for display and other operations, (‘v’ – to visit) ‘o’ – visit the current bookmark in another window, keeping the bookmark list open ‘C-o’ – switch to the current bookmark in another window ‘r’ – rename the current bookmark

Revisit YASnippet later

check this link

TODO Learn to integrate Org-mode & AucTeX mode for daily writing/brainstorming

The following paragraphs can be found here

I think outlining and drafting is best done in Org-mode. AUCTeX is best later in the process.

When drafting you can use Org-mode to outline your document. This is basically to make some headlines that represent content you want to include in your document and then you work by filling in content under each headline. The headlines are foldable so that you may get an overview of your document's structure. You can also mark the headlines with "DONE" or "TODO" to track your progress. See for instance this tutorial for details.

Once you are happy with a draft you can use Org-mode's export function to export the document to LaTeX (you can export from Org-mode with a preamble of your choice). I have written about this in detail in another answer. Then you can fine-tune the document with AUCTeX. Note that AUCTeX also has an outline feature for getting an overview of the document.

The idea for this approach is to take advantage of Org-mode's functions that eases the process of recording, organizing and developing ideas and also benefit from AUCTeX many functions that helps in editing LaTeX files. I believe that this is a good approach but it requires the user to know the basics of Emacs, Org-mode and AUCTeX. Fortunately there are good tutorials and manuals for them. One just need to know the basics, which are not hard to learn, to start out. Then one can learn the rest as one goes. For how to learn Emacs and some more notes on workflow see A simpleton's guide to (...)TeX workflow with emacs.

TODO think about the global structure of the thesis

2.9 Aha! Insight

2.10 Kata challenge & efficient coding

2.11 first discovery of gravitational wave

the pronunciation of the

通常发本音，只有在后面的单词是元音开头时才发 thi，元音的话就要自己看书啦，不过基本上什么 a,o,u,e 开头的大部分都是元音开头的单词。另外强调的时候也可以发 thi , 还是那句话，专业。另外，搞笑时一般也发 thi

2.15 install mysql on windows 10, starting to learn SQL

2.19 唯有大慈大悲手段，方能做出断爱等事

如何评价弘一法师「遁入空门而抛妻弃子」的行为？

作者:王路链接:https://www.zhihu.com/question/21041918/answer/21034906 来源：知乎

原先看过一个俞敏洪的访谈，访谈里对弘一法师推崇备至。推崇完了又感慨：他老婆带着女儿在寺庙门口哭，他就是不出来，换我肯定做不到。

这么做是残忍。但成佛需要这种残忍。这种残忍不是对别人残忍，更是对自己残忍。圜悟克勤禅师说，要有杀人不眨眼的手脚，方可立地成佛。李叔同就有这等手脚。

当年李叔同和夏丏尊在同一所学校任教，夏丏尊是舍监，宿舍丢了东西，夏丏尊问李叔同如何是好，李叔同想出了一个办法：“你贴出告示请偷盗者出来，说倘偷盗者三天不出来，我作为舍监监管不力，当自杀谢罪。”李叔同又说，不过，三天之后若偷盗者不出来，你必须自杀。夏丏尊想想还是放弃了。用佛教里的话说，李叔同天生有佛缘，这或是得益于过去无量劫来的修行和亲近善知识。至于残忍，也是不得不残忍。在佛家看来，所有的烦恼和痛苦，归根结蒂只有两个原因：无明，有爱。断爱，是断除烦恼的第一步。世俗之人眼中的残忍，佛家眼里恰恰是慈悲。因为“爱别离”与”求不得“之苦，是贪著“爱取有”而生的业果。有爱便有苦。

即便是从世俗义谛上来说，如果你不再爱一个人，那个人又深爱你，你不予割舍才是对她最大的残忍。* 因为你是有能力主宰的一方，而她绝无主宰自己的力量 *。倘你不予割舍，便是给她机会造作新的业，再遭受更多的苦果。李叔同不再见妻子，只是截然斩断过去，不给彼此再造作新业的机会。唯有大慈大悲手段，方能做出断爱等事。

2.21 plan to do leetcode problems

advise from Zhihu

- 1 推荐流程 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。作者：Zhang Justin 链接：https://www.zhihu.com/question/26243279/answer/32459565 来源：知乎

  leetcode 蛮有用的，我当时刷了 80%，感觉对所有面试题都能短时间给一个大概的答案。没在国内找过工作，所以不好说。大家干这行的，如果想换工作，都会去刷题，因为面试能力和你平时的工作能力其实差别挺大的。你表示自己能熟悉各种 machine learning，随意实现 dbn，rbm，q-learning 蒙特卡罗，可一段时间不刷题，一下碰上写个 thread-safe circular linked-list 还真一下反应不过来。。。我觉得题主提问是求面经，而且是问我们寻常人该怎么准备面试，也不是来看几位大牛吹水的。我就分享下自己找工作时候的一些准备过程。

  认真看了一本算法书，整理了一下自己的算法体系，像算法导论，或者 Algorithm Design by Jon Kleinberg 也挺不错的。花一周认真看看，如果你连几个 search，tree 都不熟的话还是多花点时间。。好处是你给自己建立了一个自洽的算法脉络，以后看面试题，或者处理其他一些现实问题，你都会不由自主的去把问题抽象化，然后归类，我自觉这种思考方式使我受益良多。

  看一下 Cracking the code interview. 这本书大家应该都说烂了，但这本书长盛不衰必然也是有理由的。它的题量不多，书里废话也挺多。但它的好处是，它做了一个整理面试题然后归类的过程，而且基本把你可能遇到的所有面试题类型都覆盖了。然后开始刷

  leetcode 吧，从简到难，认真刷一遍。建议不要用任何 ide，打开没有 error check 的 text editor（emacs，vim，notepad++ etc)

  最好在有几个可以吹水的领域。因为面试不是一个单纯做题的过程，对于我们平凡人而言，既然不能一直秒杀面试官，那就想得正常点，和面试官交个朋友。而做技术的，大多数都乐意结交那些知道一些自己所不懂领域的人。我记得我面的时候，还给人划过 Android 整个 system stack，从 dalvik 画到 kernel，还有人跟我聊在 microkernel system design 之类的。

  保持一个积极的心态。我知道知乎有很些大神谈面经就是“我从来没被拒过“，但你要相信，这世界上还是平凡人居多的。我的朋友，包括我自己，在面试中都是一个越挫越勇的过程。像我一个朋友，他被 google，twitter，apple，microsoft 都拒过，但最后去了 Facebook，还有一个朋友被 google，square 拒过，最后拒了 amazon 去了一家 startup，现在已经被某大公司收购，走上高富帅之路。我被 facebook，amazon

拒过，google intern 拒过一次，但 microsoft 就走的很顺，后来 google 很奇葩的把我面试时间 schedule 错了，我过了第一轮，签了 MS 就不再面了。所以无论遇到什么困难，都要保持乐观，你不是一个人，平凡人也有很多，要相信自己。

- 2 按 tag 刷题 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。作者：Lou 乎链接：`https://www.zhihu.com/question/36738189/answer/69420404` 来源：知乎

  按照 tag 刷起来挺好的。

  举个例子：比如复习链表的内容，就选 Linked List 这部分的 23 个题目。刷完之后可以再总结一下常用的方法和数据结构构造方式。总之不是为了刷题而刷题啊，是为了弥补一部分的知识去做。

2.22 The Devils are often in the Details, SICP is really a fantastic book!

backward jump to higher order heading in Org Mode

Use "C-c C-u"

The Devils are often in the Details

So, get your hands dirty!

Less is more + " 奔" + " 浅尝辄止"

把这三种基本态度铭记于心，时时映照。

SICP example of the picture language

虽然有些 detail 还需再深入思考，但目前所体会到的东西已经给我一种醍醐灌顶，眼界大开的感觉。

- my thoughts

  1. The design with different layers of languages not only powerful software

  engineering idea, but can also serve as a general philosophy and be applied to daily life.

  1. How to judge/compare the representation power of a/two language/languages?

2.26 I still remember some expressions of math analysis teacher — professor Bahri Abbas

budge 让步；改变主意

Abbas Bahri

- my memory for Bahri, my dear teacher Unscrew your mind! Free yourself! Crystal Clear. Train to see abstract things. Never take anything for granted. "You should do it, not say that it is easy." — Bahri's remark on my homework The freedom is here. Always have an idea where you want to go. Start with an idea, and write your proof to the end. Just try to do that far. Draw sth, feel sth, translate what you feel. Genius + Hard Work The hidden thing Math is an experimental science, and definitions do not come first, but later on. "We guide by intuition and march the road to our destination with the help of **rationism**" When you are delighted by some intuitions, then finally try to calm down and ask :"give an example to illustrate".

- my comments on Reddit "I had him for math analysis 4 years ago. He is really an enthusiastic teacher and I would always remember his emphasis on "Crystal Clear" mathematical thinking."

- an interview I found on the Internet

    1. Tell us about yourself and your specialty or research I am a mathematician, born in Tunisia. I have four children. My wife is from Italy. My area of expertise is Non-linear Analysis…..

        (a) What were some of the milestone junctures of your life as a scientist?

        I can see two of them: the time I spent in Paris, France, listening to talks at the College de France, within the Analysis and Partial Differential Seminar of Jacques-Louis Lions (who inherited the seminar from Jean Leray); the time that I spent with my students at my home in Tunisia, where I used to spend six months every year in the nineties, laying a foundation for a school in Non-linear Analysis.

        (a) What are the most important areas you are working on nowadays? Contact Form Geometry, Conformal geometry, Yamabe-type problems.

(b) Where do you see science should opt to? i.e., which direction, it should direct its attention to? I do not know.

(c) Do you have/had role model scientists? No.

(d) Some say: that after the unique generation of Einstein, Schrödinger, Planck, Bohr & others, science was not able to come with breakthroughs. Do you agree? Explain pls. I disagree. Breakthroughs have various dimensions; it takes a lot of time and a lot of patient work of generations of scientists to come to turning points in the thinking.

   i. Is philosophy for scientists a luxury or a necessity?

   A hobby and a necessity for me.

(e) What is the societal role of scientists? Are we doing enough or we are lagging behind? I do not know.

(f) Where the emphasis of developing countries (Third world) should be: Solving the immediate problems or indulging in basic research? Both.

(g) Are distinguished scientists made or they are just born? Made essentially.

   i. What are some of the most fascinating discoveries scientists have made in your area during the last 20 years.

The understanding of non-compact phenomena.

(a) Are you optimistic about the role of science & scientists in contributing to world problems such as dwindling economies, wars, famine, abuse of natural resources, polluted environment? Yes

(b) Some science education specialists say the bad image of science in the eyes of public has to do partially with inappropriate curriculum; i.e. content-driven instead of context-driven and this is the result of our focus to prepare students for general exams such as SAT, GRE, International Olympics$\cdots$ etc? What do you think? I do not know.

2.28 计划开始增肌减脂

March

3.1 Latex Tutorial by Indian user group

京口

京口是六朝长江下游军事重镇。京口是镇江古称，西周时属宜的封地，“宜”是吴和吴文化的发祥地之一，春秋时京口时属朱方邑，后朱方改谷阳。至秦始皇三十七年（公元前 210 年），秦始皇东巡会稽，途经京岘山见有王者气，命三千赭衣徒凿断龙脉，以败王气，故改命丹徒县。京岘山西北有雄伟的北固山，那个时代它的后峰伸入江中，北固山的后峰、中峰、前峰起伏连绵，前峰环抱着开阔高平地块，古人把前峰一带称之为京，取义为《尔雅》的“丘绝高曰京”，口指北固山下的江口。东汉末年，孙权称霸江东，于公元 209 年将苏州的根据地迁至京口，在北固山前峰筑铁瓮城，号称“京”通称京口。

Latex Tutorial by Indian user group

Read the **cross reference**, **floats(table & graphics)** chapters today. Really illuminating reading experience.

学习 emacs 能很好的提高工作效率吗？

作者：子龙山人链接：`https://www.zhihu.com/question/38484391/answer/88845653` 来源：知乎

我又要来安利一发我录制的视频了: `https://github.com/zilongshanren/Spacemacs-rocks/issues/5`

作为一个 5 年 vim，2 年 emacs 用户，我来谈谈我的一点看法吧。楼主怀疑学会 emacs 是否真的有助于提高工作效率。相信很多人都会有此一问，甚至一些已经有好几年 emacs 经验的用户。因为 hack emacs 实在是太有诱惑力了，每当我成功 hack emacs 之后，我都非常有成就感，这不是解决一两个 bug 所能带来的，这也是 emacs happy hacking 的魅力。但是有时候 hack 真的是很费时间，可能不知不觉一个小时就过去了。所以一定要避免在工作时间去调教你的编辑器。就像其他回答所说的，你利用别人打游戏的时间折腾是非常划算的。折腾 emacs 绝对是 time killer。另外我觉得一定要融入社区，不要蒙头瞎折腾，像我现在几乎不折腾了，因为我用 spacemacs。要提高工作效率关键还是对于工作时间的利用，我用 org mode 来做时间管理，目前效果还不错。工作中写代码经常用到查找文件，搜索，跨文件搜索和替换，这些 emacs 都很强。偶尔写 java 和 oc, 我也使用 IDE，登录服务器我也用 vim。这些都是工具嘛，用的顺手，能快速解决手头问题才是王道。

最后，折腾的过程才是 emacs 党最大的收获。我这里就不展开了。最后，如果你不是 geek，只用 windows，不想折腾，用 IDE 搞定一切，偶尔使用命令行，其实不折腾也行，编辑器最终也只是个工具而已。

3.2 At what level do you think/design/construct(etc.)?

At what level do you think/design/construct(etc.)?

Sometimes, this is the key question when you got stuck. Maybe it is just because you're not at the **right level of abstraction**.

3.3 insert/display images in org mode

insert/display images in org mode

if you add myimage.png, org mode will use the file myimage.png as an inline image for myimage.png. You can then run M-x org-display-inline-images to display your inline image.

adding (setq org-startup-with-inline-images t) to your .emacs will make image inlined on opening a org buffer. M-x org-redisplay-inline-images will refresh inlined image if needed.

what is the "unit time" in this particular algorithm problem?

the power of "wishful thinking"

3.4 study spacemacs documentations & relearn gnus

study spacemacs documentations & relearn gnus

- the **SPC h** key is very powerful Like **SPC h SPC**, **SPC h d f**, **SPC h d v**

3.5 Peter Liang, Zootopia

To Wu Rui Ge

吴锐哥你这问题一来就问“怎么看”这真不是一两句话能表达清楚的事情，所以那天看到你消息觉得短时间说不清就放一边了。梁警官的事件最近我没有跟踪，但总的来说我觉得他很大程度上是被纽约警局所牺牲的。因为前两年和种族歧视有关几次骚乱的发生，所有涉案警官都没有被起诉，这次给人的感觉至少是纽约警局选择性地放弃了梁警官。

在大部分美国人的印象里，亚裔属于“好好”公民，很多时候也不是特别团结地维护自生的权益，这在我看当然是偏见，但若非有无数的例子也不

会给人以这样的印象。我觉得这次梁警官的事件至少让我开始思考在美国这个地方，少数群体应不应该抱团，应该怎样抱团这些问题。以前我觉得无关紧要，现在看法有些改变。

华人不爱抱团，总爱彰显自己特别公平公正，大是大非帮离不帮亲，这在目前阶段非常不利于整个群体被接纳。这次的梁警官事件我在微信转发的意思也是希望周围的朋友一起关注这件事，做些力所能及的事情，改变很多美国人多亚裔的偏见。我个人没有用“歧视”这个词是因为我接触过很多不错的美国人，我觉得美国社会中能够理性思考的本土人还是有很多很多的。

Zootopia

3.9 coder 的成长

debugging

老鸟和新手的一个很大区别来自于 debug 的能力。其中最主要又可以从两方面看出来：

1. 从高层往底层找错。

2. 科学方法。

很多新手遇到程序执行结果不对（尤其是图形程序员），先认为是机器毛病（浮点精度、硬件故障），然后认为是驱动有错，再认为是系统有错，最后才开始排查自己的程序。其实 99% 的情况下是自己程序有错，然后那 1% 里面的 99% 是系统有 bug，再接着那 1% 里的 99% 是驱动有 bug，最后到硬件问题，已经微乎其微了。应该从高层往底层查，而不是反过来。

debug 一般来说是知道现象，但原因未知。这一点和很多自然科学的情况一样，所以完全也可以用科学的方法来：提假说 -> 根据假说做出预言 -> 做实验肯定或否定预言。对应于 debug，那就是假设是某个地方有问题，那么推断它一定会导致除了你看到的现象之外的其他现象，运行程序看你的推断是否成立。掌握这个方法后 debug 不在变成瞎找瞎试，而是有迹可循有系统可依赖的方法。

coder 的成长

用三年时间成长为顶级 Coder 是有可能的。

不信？排名第二的答案是酱紫说的，快速成为顶级 Coder，你需要的是「自律」和「坚持」。请注意，我们说的是「Coder」，而不是「Programmer」或「Software Engineer」。对于 Coder，我们是有客观评价标准的，那就是参加「编程比赛」。

楼主研究了一下所谓的「编程比赛」或「算法大赛」：除了国内的各种程序设计大赛，如百度之星、有道难题，更推荐参加国外的编程比赛网站，如

「topcoder」、「codeforeces」。这些网站上每个月都有比赛，只要你有一台能上网的电脑就可以免费参加，和全球的 Coder 竞争，赢得了比赛还有美金拿哦。

简单来说，你至少需要做到以下几点：• 进入 topCoder 的练习室，每天花 4-6 个小时练习编程。

• 制定一个计划，比如：每天解决 10 个初级问题，每周搞定 3 个中级问题、1 个高级问题。

• 如果在某个问题上困住了，马上去论坛求助。要知道，问问题体现了你的思维逻辑，问出好问题也是需要练习的。试着每周在问答网站 stack over flow 上问一个问题吧！

• 不要闭门造车，把你的代码展示给别人看，他们提出的意见会让你事半功倍，推荐免费网站 coder review。

• 学会看别人的代码，看代码其实比自己写代码更难，但是高手是可以轻易调用别人的代码的，推荐 github，可以轻松地找到海量的开源代码。

• 建议采取一些外部的措施克服拖延症，比如开通一个博客、或微博，每天在上面记下你今天做过的一件重要的事情，保证每天晚上都带着巨大的满足感入睡。

如果坚持做到以上这些，你完全成为一名顶级 Coder。需要注意的是，编程比赛关注的是算法能力；但是，要成为一名 Programmer 或 Software Engineer，你还需要项目经验。正如 Facebook 喜欢雇佣所谓的「Full stack programmer」，就是一个人从设计、到交互、html、css、javascript、server、sql、架构，以及数据统计都能做。成为 Full Stack Programmer 最好的方式就是不断做个人项目。

参加 Hackathons 或 game jams 就是锻炼项目能力的好方法。在规定的时间内，一群开发者分享、讨论、组队、分工协作，用创意思维完成项目任务。无论是制作一个游戏，还是网站，在整个项目的进程中，你会不断经历发现问题、解决问题、获得经验的过程，从而保证在实际工作中也能轻松解决问题。

总之，楼主以为，入行时间并不是衡量人才能力的绝对标准，碌碌无为的「老人」在各行业都不鲜见。在 3 年之内能否成长为一名优秀的 Coder 或 Programmer，其实取决于你对所做事情的理解程度。对所做事情理解的越深，你就会做的越好。

成为一名优秀的程序员和成为其他行业的高手一样，都需要不停地学习、练习、反省和总结。这不仅是最初 3 年的要求，而是贯穿整个职业生涯的要义。

所以，初级程序员想要获得快速成长，一定要拥有不怕麻烦的责任心，和不满足于现状的上进心。

3.12 AlphaGo defeated Lee Sedol

Google vs Facebook（田渊栋）

我半年前从谷歌 X 的无人车组跳到 Facebook 的人工智能实验室（FAIR），感触良多，这里写一些分享给大家。

虽然 F 和 G 并称一流的 IT 公司，但是其实内部是很不一样的，甚至可以说完全相反。加入 FB 之前，问过很多朋友，大家的意见综合起来是 FB 有点“乱”，没有统一的平台，各组管各组忙，代码质量比 G 差很多，文档也少。这听起来挺吓人的，但认真想想，反过来说乱才有机会。G 最大的问题恰恰是一切都井然有序，能出大成果的地方都出完了，员工就像螺丝钉，只要在自己的岗位上做好修补就行了。

F 没有五花八门的内部工具，用的大多是开源的略作修改，所以 F 的员工流动性非常大，因为工具和平台熟悉了，到外面一样可以用。举个例子，我半年前加入，现在已经比三分之一的员工来得早了。相比之下 G 的内部工具太多，待久了习惯了这些工具，再加上成年累月的股票，往往跳走不易。我们组的分布式平台当然没有 G 的复杂，还有好多地方做得不好，然而正因为简单才容易理解，也容易给它动手术学到东西。那个写平台的人，往往就是坐在邻座的邻座的的工程师，哪天需要加新功能了，打个招呼就行了，甚至自己可以动手写完让他审核。要是在 G 碰到这样的问题只能举双手投降，因为那是远在校区另一头某个人某个组的事情。

F 太多项目可以做，合作也很容易，往往是吃顿饭大家聊聊就开干了。而在 G 大家抢项目做，员工并没有太大的自由度，一般指望老板还有老板的老板和别的组抢。我加入 F 这半年，在没有提前通知自己老板的情况下，单独和产品组的成员合作了一个项目，思路简单但是非常有用，单机处理 100G 数据，花了三个月发布了，还上了 VP 和 Engineer Director 当月的 Top List，备受大家关注，这在 G 是完全不可想像的。

作为研究员，我再也不用为代码格式和变量名字伤脑筋，再也不用因为别人没来及审核自己的代码而无法工作，因为存在一个实验目录，可以不经审核提交代码。G 对代码的严格对公司是有好处的，但它太过死板的规定经常限制发挥。我有时真弄不明白，为什么在两三天后审核回来时，还要改变量名改接口设计呢？我分明已经在这个基础上写了四五个扩展了，再花时间改回去，真心是很伤士气和进度的。

这半年说话比原来多了。作为码工只要和同事把现有的平台搞清楚，然后照着写程序就行了；但作为研究员需要不停地和各种人交流，同事，访问教授，实习生，理解他们的工作，提出意见和建议。你不愿意说话，别人（甚至 Yann）会找你说话，久而久之就发现，自己永远得要对自己的工作，即便是正在进行中的工作，有一番概括和推广，自己的工作得要自己吹起来，别人才会知道并且欣赏，长远来说，这对个人的发展更有利。

我们组做深度学习主要用 torch/lua。一开始学习曲线比较陡峭，但用惯

了则相当好用。这个语言有趣的地方是，它很简单，简单到你可以理解语言设计的每个细节，对每个出现的 bug 都会有相对深刻的理解。我曾帮过我们组的工程师设计一个特殊的用来做 profiling 的 lua 类，身为研究员还可以在设计上帮到工程师，是挺自豪的。

我加入之后，经常从早上十点干到晚上八九点，然后锻炼完之后回家继续忙，经常周末加班，带几个实习生再加上做自己的研究。看起来工作强度是更大了，但为了自己的事业，有什么不可以的呢？最重要的原因是，相比在无人车组一心二用，白天做应用晚上自己折腾研究，我的内心深处，还是想集中精力花几年去理解深度学习本身，做一做它的理论框架。我的老板也非常支持这样长期的研究，这个是 FAIR 真正让人心动的地方所在。

这两天绩效考核，我老婆评论说我这半年干的事情比在 G 家一年还多，有产品发布也有研究，我想这就是真正把兴趣用在工作上的结果。我还记得自己最后一天在 G 的日子，HR 小姑娘最后问我为啥离开，是不是因为 X 的工作太辛苦，需要一些工作和生活的平衡？我笑了笑，敷衍了几句，心里想起了《冰与火之歌》里的那一句台词——

雪诺，你什么也不懂。

2015-07-14

3.13 Lee Sedol strikes back on my birthday!

小李，人类的胜负师

这几天真是一个跌宕起伏的故事，从轻敌到惨败，再到不屈不挠的反杀，昨天天才少年的落寞，和那一句，这是我李世石的失败，不是人类的。到今天战胜 alphago 狗，不动声色的复盘，这就是一个人类可以有的全部自尊，不屈，才能，冷静。可以想象以后会有许多以这为原形的文学吧。是的，我们从未放弃。

作者:张三链接:https://www.zhihu.com/question/41325834/answer/90532996 来源：知乎

虽然是老生常谈，还是忍不住感叹一句。处于上风的时候（做顺手的项目，交佩服自己的朋友，调戏不爱的姑娘）浑身的机灵劲儿怎么抖怎么有根本不是本事。像李世石那样也得意过也张狂过，但是在最不得意最不张狂，被虐得体无完肤信心体力都落在最低点的时候，还能咬着牙槽反戈一击，才算是真男人啊。— 木遥

Run time, compile time, link time & load time

- how to avoid run-time error The following guidelines show how to avoid run-time errors. These are only guidelines, as the conditions under which run-time errors occur are dependent on the context of your application.

If, for example, you use the GET function to locate a record, you must be prepared handle the possibility that a run-time error can occur if there are situations where no record is found. If you are certain that the specific context precludes this situation, you can omit handling a possible run-time error. (The context could be that the existence of a record is verified before the GET function is used.)

There are two categories of run-time errors:

1. Errors that are related to the use of data types

2)Errors that occur if a function does not succeed in doing what it is supposed to do

Division by zero does not fit into either of these categories, but it has been placed in the first one.

You can only prevent some errors (mainly related to data types) from occurring. Other errors cannot always be avoided, but you can write code that shields the user from the error. Instead of the default error handling (which displays a message, closes the form that was active when the error occurred, and rolls back any changes to the database), you can write an error handler that, for example, gives the user an opportunity to correct the input that caused the error. This error will display a message that explains why the error occurred.

3.16 Have some idea about the vast land beyond SICP after read a tutorial introduction to Lambda Cal by Raul Rojas

vast land beyond SICP and current thoughts

各种各样的类型推导，想懂得多一点的话已经进入数理逻辑和代数里的范畴论。其中的一种，即 lambda calculus，就已经是 functional programming 的基石。

我很有兴趣，但我也明白现在这个阶段没有必要也没有时间深入了解，所以大概能通过深入阅读 SICP 理解 functional programming 的原理并能根据原理写一些有用有意思的程序就够了。以后再图深入 Programming Language 这个领域。

我觉得今天阅读 SICP section 3.1 以及 lambda cal tutorial 最大的体会就是————怎样去定义语言中的“替换法则”确实很 subtle，而如果想在语言中引入赋值，那变量环境/域以及相关 scoping rule 就不可避免要准确来定义了。

向热爱计算机科学的你推荐 SICP(转)

先谈谈关于《计算机程序的构造和解释》（后面简称为 SICP）的几个八卦。

本书曾经是 MIT 本科第一门课的教材。前两年被 Python 取代，在 geek 中引发了轩然大波。有兴趣可以 Google 一下 [sicp mit python]。本书在 Amazon 上的评分严重两极分化，五星（>90）和一星（>50）为主，彻底反正态分布。本书在 Amazon 上排名最高的书评出自 Peter Norvig，当然是强烈推荐，顺便狠狠地鄙视了给一颗星的同学们；第二篇出自 Paul Graham，还是强烈推荐。本书别名紫书（The Purple Book），巫师书（The Wizard Book），或者干脆 The Book。

这是一本什么样的书？

前言说，这是一本给 MIT 学生的入门级（entry-level）计算机科学教材。作者的出发点有两条：

语言首先是写给人看的，只是恰巧（incidentally）能够运行。这当然是个修辞，格外强调代码之可读。语言的语法，漂亮的算法，数学的分析，这些统统都不重要。最打紧的是如何控制复杂度（The techniques used to control the intellectual complexity of large software systems）。

在本书成书的年代（1984），以上言论即使不算正邪不两立，也够的上离经叛道了。

通俗的说，这本书教你如何用最基本的构造和原则，解决复杂和多样的问题。用摄影打比方，这本书不比较尼康和佳能，不介绍繁杂的机型和参数，不介绍后期处理的技巧。这本书只讨论光线、色彩和构图，以及如何在不同场景拿捏这些基本原则组合出美妙的照片。

这本书适合初学者吗？

不好说。Amazon 上的一颗星评价大多鄙视本书已经过时或者太过高深。我个人看法，它很适合一部分初学者，但是需要满足几个条件：

热爱计算机科学有时间和耐心受过（高中水平）数学和抽象思维的训练

所以，如果只是想领一份程序员的薪水，这本书完全可以略过。并不是说这本书有多么不实用，只是计算机科学与写代码并不是一码事。

这本书广而不深，讨论到了非常多重要的思想，有些甚至冷不丁出现在注释里（比如 Y 算子）。内容安排很照顾初学者，循循善诱；语言直白简单；代码大多简明自然。

至于习题，个人认为只要认真思考，大部分都不是很难，需要耐心多于智力。有时间不妨多做几道。这本书适合有经验的程序员吗？

还行。如此庖丁解牛般的讲解，其他书中不多见；内容简单，思想却不过时。另外，国内绝大部分程序员都从命令式语言入门，不妨接触一些函数式思想，开开眼界。如果时间不多，至少看看前两章，学习一些解决复杂问题和编写优雅代码的技巧。

为什么我们要学习这本书？因为这本书告诉我们如何抽象。为什么我们

要学习如何抽象？因为抽象是我们控制软件复杂性的重要手段。软件是人类有史以来最复杂的系统。其一、软件系统本身规模庞大，参与人手众多，难以管理；其二、环境和需求不断变化，且错误难以避免。

人类无法驾驭过于复杂的事物，于是只能寻找方法简化软件系统：把系统分为许多子部分，人们开发一个部分的时候，系统其他部分都是一种抽象，无需了解其细节。

本书讨论的就是系统的组织和设计，有哪些方法可以帮助我们控制软件的复杂度。Scheme 好学吗？

其实 Scheme 是一门异常简单的语言。直来直去，除了括号多，基本没有旁门左道（比如指针）和撕心裂肺的语言构造（比如模版、多重继承、Monad）。再者，这本书的内容和具体的语言基本没有关系，思想才是重头戏。如果实在有顾虑，推荐先读两本薄薄的小册子：The Little Schemer 和 The Seasoned Schemer。这本书到底讲什么？

本书按照内容可以分为三个部分：过程抽象（第一章）；数据抽象（第二、三章）和语言抽象（第四、五章）。

过程抽象部分比较简单，先介绍了 Scheme 的基本语法，让读者初步领略函数式编程的风采。对于有一定编程基础（相信国内极少有人入门就读这个）的读者来说，会有耳目一新的感觉，原来递归和迭代可以有另一种表现形式，但并不难理解。习题也比较简单，不会用掉太多的时间。过程抽象的概念也很简单，就是编程语言中的函数，目的是封装计算过程的细节。关于何时应该用过程抽象的原则是：一切可以定义为过程的计算片段都应该定义为过程。

数据抽象是我认为的本书的核心，也是最值得我们仔细研读的部分。关于数据抽象最直接的理解就是面向对象编程，如 C++，而 Java 和 C# 则是更彻底的数据抽象。把一组过程抽象（类的方法）集中考虑，并加入内部状态（类的变量），就是一个数据抽象。每个数据抽象都应该把自己的内部对象状态和对象的实现隐藏起来，对外通过一组接口进行消息传递。这样听起来好像本书与一般的面向对象书没有区别，但实际上，这些都是我自己的总结，书里面不会把这些概念直接罗列出来，而是通过一个个巧妙的例子，让读者一步步深入，感叹原来 A 还可以这样抽象，原来 B 还可以这样封装。个人认为如果时间有限，读完前三章已经可以领会本书大部分思想了，后两章可以不读。

语言抽象是指自己发明一门语言，以解决某一特定应用领域的问题。在这一领域中，自己发明的语言会比其他通用语言更方便。定义了新语言的语法后，就要自己去实现该语言的编译器或解释器，可以通过现有的语言去构造。这一部分包含了许多编译方面的知识，但又与编译原理中的构造方法有不少区别，自己看书很容易看得云里雾里，听老师讲课才好一些。大部分习题很难做，一部分习题非常难。

第一章讨论程序设计的最基本原则：原语（primitive expressions）、组合（means of composition）和抽象（means of abstraction），以及如何利用这些

基本原则化解复杂度。重点是过程抽象和高阶过程（high-order procedures）。本章的例题十分精彩，抽象和组合的过程十分清晰。有关递归和迭代的讨论也非常耐读。

第二章讨论数据抽象，即利用基本数据构造复杂结构。Scheme 里的基本构造能力只有 cons，但由此可以组合出所有实用的结构。图像语言、符号运算、集合表示、哈夫曼编码和复数系统都是经典实用的例子。顺带还介绍了 data-directed 方法，与面向对象中的封装有异曲同工之妙。

即使没有太多时间，我觉得前两章也值得值得细读。尤其是例子。

第三章主要讨论了状态（local state）和环境（environment model），可变数据结构（mutable data），以及状态和时间的交互（concurrency 和 laziness）。前两章用到语言是 Scheme 的一个没有副作用的子集，从这一章开始涉及解释器的核心机制，尤其是状态的管理，及其优缺点。

第四章用 Scheme 实现了一个简单的 Scheme 解释器。重点是讨论语言的解释过程，以及如何针对问题（领域）创造和修改语言，从中可见 DSL（Domain Specific Language）的思想。后三节各自讨论一个工程中不常见但高效解决特定问题的语言变种及其实现。

第五章介绍将 Scheme 编译为现实中的寄存器机器模型（register machine）。重点不是编译技巧（Scheme 压根不需要文法分析），而是基本构造（条件、过程，等等）对应于寄存器模型的实现。略带讨论了最简单的垃圾回收。

后三章较深，最好略有一点语言、编译和体系结构的基础，或者多些耐心。语言会影响思维

如果要问现代数学最重要的概念是什么，那毫无疑问就是函数了，或者更确切地说，是映射。泛函这个词，或许对非数学系的同学来说有些陌生，但如果写成英语 functional, 看起来就眼熟多了。狭隘一点地说，泛函就是以函数为参数，返回值是数值的一类函数。看到这里相信不少同学都发现了，这就是在很多语言中被称为高阶函数（high-order function）的那个东西。泛函在数学中是如此普遍的概念，现代数学几乎无处不会用到。数学家们很自然地在集合上添加运算，构造空间；从一个空间映射到另一个空间，创造泛函。对泛函做变换，构造泛函的泛函等等。

为什么我要在这里提到数学和泛函？因为在我看来，lisp 是一门以表达数学为己任的语言。正如 SICP 中希望表达的一种观点：语言会影响思维。如果数学推理过程中最频繁应用到的泛函，在计算机语言中却没有对应的表达，换言之数学思维不能很自然地表述为计算机语言的话，那么计算机对于数学研究的意义就显得很可疑了，毕竟那时候的计算机可不是用来玩大菠萝 3 的。所以这里就有了两拨人，务实的一拨人开发出了 fortran ，力主解决数值计算；务虚的一拨人则创造了 lisp ，试图一举解决符号计算的难题。在 John McCarthy 所作的 history of lisp 中这样写到：

Then mathematical neatness became a goal and led to pruning some features from the core of the language.（保证数学上的简洁性成为我们的目

标，并因此拒绝了将一些特性加入到语言核心中。）

This was partly motivated by esthetic reasons and partly by the belief that it would be easier to devise techniques for proving programs correct if the semantics were compact and without exceptions.（这部分是基于美学上的考虑，部分是因为我们相信，紧凑而没有特例的语法才更有可能设计出一种从数学上证明程序正确的方法。）

之所以讲了这么多关于数学和历史的东西，是因为我觉得在看这本书前，最重要的是理解：lisp 是什么。而我又一直相信理解一样事物最好的办法就是理解其历史。（顺带说一句，以上历史都是在看过书以后才找的，所以也是我的血泪教训……）如上所示，lisp 是一门为了表达数学推导过程而诞生的语言，所以不可避免地使得 SICP 前两章的例子几乎全是数学问题。代码只是其形，而其神是纯粹的数学。所以这里似乎就陷入了一种两难的境地：如果执意于写代码的话，那看起来做的都是形而下的工作；而如果只思考问题的数学原理的话，那姑且不说是舍本逐末，至少也是偏离主题了。在看完 SICP 以后我始终怀着这种疑问而不解——看的时候是不会有这种感觉的，因为注意力全部纠结于书中的题目了——不过在写这篇书评时又翻了一下第一章，似乎明白了。

小节 1.1 写到：

一个强有力的程序设计语言，不仅是一种指挥计算机执行任务的方式，它还应该成为一种框架，使我们能够在其中组织自己有关计算过程的思想。每一种强有力的语言都为此提供了三种机制：基本表达形式，组合的方法，抽象的方法。

所以我认为 SICP 这本书最主要的目的，就是“教你用 lisp 的语言，来组织，来抽象，来表达想法”。从这个意义上来说，SICP 和一本 Learning Python ，或者一本 C Programming Language 并没有太多的区别，依然讲授的是“用特定的语言求解特定的问题”。不过略有不同的是，lisp 太特殊了，导致从 c 转向 python 或许不需要太多的思维转换，但从 c 转向 lisp 却需要对思维习惯大改造一番，这我想就是 SICP 地位如此之高的原因吧。我也同意，学习 SICP 确实很锻炼思维，以及培养一种更加高度的抽象习惯。其实在看 SICP 的过程中，很多时候我都会感慨，“如果我不是数学系的，这一段到底会怎么理解呢”。一个典型例子就是习题 2.6，初看我也一头雾水，后来才意识到 zero 是 f->id 的泛函，正是零映射，one 是 f->f 的泛函，正是恒同映射，也就是函数空间的 1。如果没有学过泛函分析的我来看这道题目，估计只能好不容易推导出规律后，感慨于 Church 计数的“巧妙”了。所以从好的层面来看，SICP 至少能够带来泛函的直观感受，因此我才说 SICP 是一本写给 CS 人的泛函数。但是从坏的层面上说，数学抽象毕竟是象牙塔里的产物，当好不容易抽象出一个优雅的模型却发现手头的语言难以表达或者效率上有种种顾虑的时候，还是很郁闷的吧。

前面貌似说了 SICP 的不少坏话，其实只是想拉低一下 SICP 的评价，至少使得后人不至于期待过高。SICP 是一本好书，至少是一本有趣的书，这

点我是非常赞同的。就冲着她那创意的封面图和作者头像，每章开篇都会引用一段（非常利于装逼）的名言，以及用半页的篇幅讲述和主题完全无关的 MIT 第一任校长的生平，想不有趣都难啊。不过我还是世俗一下，列一下自己看过这本书以后比较” 现实” 的收获：

对于构造递归式的训练。相信做过的都深有体会……列表处理流程，也就是 map-filter-reduce 。流处理这一节让我终于明白了 generator 的意义。从另一个角度看程序和程序设计中的问题函数式程序设计多种多样的程序组织方式丰富多彩的编程模式对一些基础问题的理解……

关于习题

最后说一下习题，习题的重要性想来大家都很清楚。

本书共有 5 章，每章都有近 100 道习题。这本书可以说是时间黑洞。每章分为 4-5 节，每节有几个小节，全书有一百小节（即 X.X.X）左右。我以小节为单位进行了估算，包括完成习题，每小节大约需要一个小时。当然不同小节难度不同，有的耗时长些，有的短些。于是读完本书并做完大部分习题需要上百个小时。再加上听课或看视频教程的时间则会更长。所以我觉得恐怕只有在校学生才有时间和精力来完成这本书的学习。

不过对于 SICP 来说，我觉得习题未必都要写成代码，在纸上写出思路和关键代码也未为不可。因为 scheme 的编码效率实在不高（就是 scheme 逼得我给 vim 装上 surround 插件……），而习题重要的还是整个抽象的过程。另外就是，要找个好一点的解释器，我下了 MIT 的 scheme 解释器发现各种操作太不人性了……为什么推荐 SICP？

向大家推荐 SICP，不知道有多少人看了，也不知道有多少人明白了，更不知道有多少人惊叹了。或者你根本不屑一顾，或者你看见 Lisp 那层层括号心生畏惧，又或者你了了一瞥，觉得没什么精彩之处。那我真的很失望。

我为什么要推荐 SICP，而且为什么如此执着？这本不算厚的书带给我的观念，是从未有过的，是关乎于软件本质的。曾几何时，我觉得我看到了计算机编程书中没有的哲学观，但这一次我的梦破灭了，那些已经被写进书里差不多快 30 年了。

我现在就来谈谈我的心得，以再次向你展现这本书的魔力。

第一章作为基础，作者并没有象后续章节写太多的软件思想，主要还是介绍 Scheme 语言，所以草草看去，没什么精辟之处。不过在第一章中，作者用了大量的篇幅来探讨数学问题，因为他想向你揭示程序设计中的核心哲学：抽象。而数学无疑是最好的例子。

了解数学史的人，应该知道整个数学史，就是一个不断抽象的历史。古希腊人将字母引入计算，使数学不再只是算术，而且具有表达抽象规则的能力。近代数学对函数和微积分的探求中，用 f(x) 替代了多项式表达式，函数更一般了，然后 n 维空间、复分析、映射、泛函，抽象代数、群论，等等等等，直到集合论，摧毁了数学的基石，使数学界再次陷入沉思。

构造程序的方法也是抽象。从最简单的元素开始，基本元素 (自演算表达式，包括数字，字符串和布尔值)，然后定义基本过程（基本运算符，四则

运算和布尔运算），进一步，自定义标识符（如同代数），再自定义过程（函数），再将过程作为值参与运算（高阶过程）。一步步的抽象，形成了整个程序的结构。而我们编程，无非就是从现实世界抽象出模型，再将模型不断的提炼抽象，属性、方法、类、继承、层次、框架。

编程就是一个不断抽象的过程。我再次把作者在第一章末写下的结论抄在这里，作为最后的注脚。

“作为编程者，我们应该对这类可能性保持高度敏感，设法从中设别出程序中的基本抽象，基于它们去进一步构造，并推广它们以创建威力更强大的抽象。当然，这并不是说总应该采用尽可能抽象的方式去写程序，程序设计专家们知道如何根据工作中的情况，去选择合适的抽象层次。但是，能基于这种抽象去思考确实是最重要的，只有这样才可能在新的上下文中去应用它们。高阶过程的重要性，就在于我们能显式地用程序设计语言的要素去描述这些抽象，使我们能像操作其他计算元素一样去操作它们。”

Perils of the Java school(Joel)

Lazy kids.

Whatever happened to hard work?

A sure sign of my descent into senility is bitchin' and moanin' about "kids these days," and how they won't or can't do anything hard any more.

“You were lucky. We lived for three months in a brown paper bag in a septic tank. We had to get up at six in the morning, clean the bag, eat a crust of stale bread, go to work down the mill, fourteen hours a day, week-in week-out, and when we got home our Dad would thrash us to sleep with his belt.” —Monty Python's Flying Circus, Four Yorkshiremen When I was a kid, I learned to program on punched cards. If you made a mistake, you didn't have any of these modern features like a backspace key to correct it. You threw away the card and started over.

When I started interviewing programmers in 1991, I would generally let them use any language they wanted to solve the coding problems I gave them. 99% of the time, they chose C.

Nowadays, they tend to choose Java.

Now, don't get me wrong: there's nothing wrong with Java as an implementation language.

Wait a minute, I want to modify that statement. I'm not claiming, in this particular article, that there's anything wrong with Java as an implementation language. There are lots of things wrong with it but those will have to wait for a different article.

Instead what I'd like to claim is that Java is not, generally, a hard enough programming language that it can be used to discriminate between

great programmers and mediocre programmers. It may be a fine language to work in, but that's not today's topic. I would even go so far as to say that the fact that Java is not hard enough is a feature, not a bug, but it does have this one problem.

If I may be so brash, it has been my humble experience that there are two things traditionally taught in universities as a part of a computer science curriculum which many people just never really fully comprehend: pointers and recursion.

You used to start out in college with a course in data structures, with linked lists and hash tables and whatnot, with extensive use of pointers. Those courses were often used as weedout courses: they were so hard that anyone that couldn't handle the mental challenge of a CS degree would give up, which was a good thing, because if you thought pointers are hard, wait until you try to prove things about fixed point theory.

All the kids who did great in high school writing pong games in BASIC for their Apple II would get to college, take CompSci 101, a data structures course, and when they hit the pointers business their brains would just totally explode, and the next thing you knew, they were majoring in Political Science because law school seemed like a better idea. I've seen all kinds of figures for drop-out rates in CS and they're usually between 40% and 70%. The universities tend to see this as a waste; I think it's just a necessary culling of the people who aren't going to be happy or successful in programming careers.

The other hard course for many young CS students was the course where you learned functional programming, including recursive programming. MIT set the bar very high for these courses, creating a required course (6.001) and a textbook (Abelson & Sussman's Structure and Interpretation of Computer Programs) which were used at dozens or even hundreds of top CS schools as the de facto introduction to computer science. (You can, and should, watch an older version of the lectures online.)

The difficulty of these courses is astonishing. In the first lecture you've learned pretty much all of Scheme, and you're already being introduced to a fixed-point function that takes another function as its input. When I struggled through such a course, CSE121 at Penn, I watched as many if not most of the students just didn't make it. The material was too hard. I wrote a long sob email to the professor saying It Just Wasn't Fair. Somebody at Penn must have listened to me (or one of the other complainers), because that course is now taught in Java.

I wish they hadn't listened.

Think you have what it takes? Test Yourself Here! Therein lies the

debate. Years of whinging by lazy CS undergrads like me, combined with complaints from industry about how few CS majors are graduating from American universities, have taken a toll, and in the last decade a large number of otherwise perfectly good schools have gone 100% Java. It's hip, the recruiters who use "grep" to evaluate resumes seem to like it, and, best of all, there's nothing hard enough about Java to really weed out the programmers without the part of the brain that does pointers or recursion, so the drop-out rates are lower, and the computer science departments have more students, and bigger budgets, and all is well.

The lucky kids of JavaSchools are never going to get weird segfaults trying to implement pointer-based hash tables. They're never going to go stark, raving mad trying to pack things into bits. They'll never have to get their head around how, in a purely functional program, the value of a variable never changes, and yet, it changes all the time! A paradox!

They don't need that part of the brain to get a 4.0 in major.

Am I just one of those old-fashioned curmudgeons, like the Four Yorkshiremen, bragging about how tough I was to survive all that hard stuff?

Heck, in 1900, Latin and Greek were required subjects in college, not because they served any purpose, but because they were sort of considered an obvious requirement for educated people. In some sense my argument is no different that the argument made by the pro-Latin people (all four of them). "[Latin] trains your mind. Trains your memory. Unraveling a Latin sentence is an excellent exercise in thought, a real intellectual puzzle, and a good introduction to logical thinking," writes Scott Barker. But I can't find a single university that requires Latin any more. Are pointers and recursion the Latin and Greek of Computer Science?

Now, I freely admit that programming with pointers is not needed in 90% of the code written today, and in fact, it's downright dangerous in production code. OK. That's fine. And functional programming is just not used much in practice. Agreed.

But it's still important for some of the most exciting programming jobs. Without pointers, for example, you'd never be able to work on the Linux kernel. You can't understand a line of code in Linux, or, indeed, any operating system, without really understanding pointers.

Without understanding functional programming, you can't invent MapReduce, the algorithm that makes Google so massively scalable. The terms Map and Reduce come from Lisp and functional programming. MapReduce is, in retrospect, obvious to anyone who remembers from their 6.001-equivalent programming class that purely functional programs have no side effects and are thus trivially parallelizable. The very fact that Google in-

vented MapReduce, and Microsoft didn't, says something about why Microsoft is still playing catch up trying to get basic search features to work, while Google has moved on to the next problem: building think Microsoft completely understands just how far behind they are on that wave.

But beyond the prima-facie importance of pointers and recursion, their real value is that building big systems requires the kind of mental flexibility you get from learning about them, and the mental aptitude you need to avoid being weeded out of the courses in which they are taught. Pointers and recursion require a certain ability to reason, to think in abstractions, and, most importantly, to view a problem at several levels of abstraction simultaneously. And thus, the ability to understand pointers and recursion is directly correlated with the ability to be a great programmer.

Nothing about an all-Java CS degree really weeds out the students who lack the mental agility to deal with these concepts. As an employer, I've seen that the 100% Java schools have started churning out quite a few CS graduates who are simply not smart enough to work as programmers on anything more sophisticated than Yet Another Java Accounting Application, although they did manage to squeak through the newly-dumbed-down coursework. These students would never survive 6.001 at MIT, or CS 323 at Yale, and frankly, that is one reason why, as an employer, a CS degree from MIT or Yale carries more weight than a CS degree from Duke, which recently went All-Java, or U. Penn, which replaced Scheme and ML with Java in trying to teach the class that nearly killed me and my friends, CSE121. Not that I don't want to hire smart kids from Duke and Penn – I do – it's just a lot harder for me to figure out who they are. I used to be able to tell the smart kids because they could rip through a recursive algorithm in seconds, or implement linked-list manipulation functions using pointers as fast as they could write on the whiteboard. But with a JavaSchool Grad, I can't tell if they're struggling with these problems because they are undereducated or if they're struggling with these problems because they don't actually have that special part of the brain that they're going to need to do great programming work. Paul Graham calls them Blub Programmers.

It's bad enough that JavaSchools fail to weed out the kids who are never going to be great programmers, which the schools could justifiably say is not their problem. Industry, or, at least, the recruiters-who-use-grep, are surely clamoring for Java to be taught.

But JavaSchools also fail to train the brains of kids to be adept, agile, and flexible enough to do good software design (and I don't mean OO "design", where you spend countless hours rewriting your code to rejiggle your object hierarchy, or you fret about faux "problems" like has-a vs. is-a). You need

training to think of things at multiple levels of abstraction simultaneously, and that kind of thinking is exactly what you need to design great software architecture.

You may be wondering if teaching object oriented programming (OOP) is a good weed-out substitute for pointers and recursion. The quick answer: no. Without debating OOP on the merits, it is just not hard enough to weed out mediocre programmers. OOP in school consists mostly of memorizing a bunch of vocabulary terms like "encapsulation" and "inheritance" and taking multiple-choice quizzicles on the difference between polymorphism and overloading. Not much harder than memorizing famous dates and names in a history class, OOP poses inadequate mental challenges to scare away first-year students. When you struggle with an OOP problem, your program still works, it's just sort of hard to maintain. Allegedly. But when you struggle with pointers, your program produces the line Segmentation Fault and you have no idea what's going on, until you stop and take a deep breath and really try to force your mind to work at two different levels of abstraction simultaneously.

The recruiters-who-use-grep, by the way, are ridiculed here, and for good reason. I have never met anyone who can do Scheme, Haskell, and C pointers who can't pick up Java in two days, and create better Java code than people with five years of experience in Java, but try explaining that to the average HR drone.

But what about the CS mission of CS departments? They're not vocational schools! It shouldn't be their job to train people to work in industry. That's for community colleges and government retraining programs for displaced workers, they will tell you. They're supposed to be giving students the fundamental tools to live their lives, not preparing them for their first weeks on the job. Right?

Card Punch – yes, I learned Fortran on one of these when I was 12.Still. CS is proofs (recursion), algorithms (recursion), languages (lambda calculus), operating systems (pointers), compilers (lambda calculus) – and so the bottom line is that a JavaSchool that won't teach C and won't teach Scheme is not really teaching computer science, either. As useless as the concept of function currying may be to the real world, it's obviously a prereq for CS grad school. I can't understand why the professors on the curriculum committees at CS schools have allowed their programs to be dumbed down to the point where not only can't they produce working programmers, they can't even produce CS grad students who might get PhDs and compete for their jobs. Oh wait. Never mind. Maybe I do understand.

Actually if you go back and research the discussion that took place in

academia during the Great Java Shift, you'll notice that the biggest concern was whether Java was simple enough to use as a teaching language.

My God, I thought, they're trying to dumb down the curriculum even further! Why not spoon feed everything to the students? Let's have the TAs take their tests for them, too, then nobody will switch to American Studies. How is anyone supposed to learn anything if the curriculum has been carefully designed to make everything easier than it already is? There seems to be a task force underway (PDF) to figure out a simple subset of Java that can be taught to students, producing simplified documentation that carefully hides all that EJB/J2EE crap from their tender minds, so they don't have to worry their little heads with any classes that you don't need to do the ever-easier CS problem sets.

The most sympathetic interpretation of why CS departments are so enthusiastic to dumb down their classes is that it leaves them more time to teach actual CS concepts, if they don't need to spend two whole lectures unconfusing students about the difference between, say, a Java int and an Integer. Well, if that's the case, 6.001 has the perfect answer for you: Scheme, a teaching language so simple that the entire language can be taught to bright students in about ten minutes; then you can spend the rest of the semester on fixed points.

Feh.

I'm going back to ones and zeros.

(You had ones? Lucky bastard! All we got were zeros.)

Bad Habits for C++ programmers

说说我见到的一些不良现象吧。有些程序员干了十多年还这样。

1. 不用 namespace。导致全局空间被污染。或组织混乱带来维护障碍。或使用不便。

2. 不使用接口隔离实现/头文件设计不当。文件间依赖过度紧密难于分离。或过于松散造成使用困难。以及包含次序耦合带来的编译问题。

3. 不使用预编译头。要么不用造成编译慢; 要么“隐藏”了一些定义，使库外部无法使用。

4. 拒绝使用 c++11 及高新的语言特性。即使编译器支持也不用 auto，不用 lambda 表达式，不用 override。

5. 对 c/c++ 盲目推崇。比如用性能问题贬低他们其实并不了解的 C# 和 Java; 比如认为所有有 GC 的语言都是解释执行的; 直到被 piapia 打脸。

6. 不会或不爱使用性能分析手段。很多人你跟他谈结构不好，他开口闭口说为了性能。然后你一问具体数据就不吭声了。变成了: 我觉得会更快。。。

7. 滥用内嵌汇编。迷信 asm 的效率和逼格。常常只是把简单问题复杂化，复杂问题天书化。

8. 不写或滥用防御代码。或不检查指针有效性。函数依赖传入参数决定是否崩溃。或用 assert 代替，崩溃后连日志输出都没有。发布后出了问题就抓瞎。或有检查，但失败后保持沉默，把问题隐患扩散到其他地方。

9. 不用或滥用 exception。或完全不用导致某些逻辑复杂。或混淆 c++ 异常与操作系统异常。或随意 catch 并忽视异常。或没有对应防御机制造成内存泄露/漏过初始化等问题。

10. 夸张的参数表。不对参数封装或抽象为对象。使得接口难用/易出错/无谓的参数 copy。

11. 盲目使用 c 的不良遗产。如随意的类型转换/类型擦除/函数参数默认值/函数变参/滥用 union 代替转换函数等等。代码建立在过多的隐喻上。

12. 滥用 const 关键字。过分强调实用 const，近乎原教旨主义。绑架别人的接口无谓提升复杂度。

13. 不了解/不愿用 stl。排序，搜索，数组，字符串都单写一套。问题多，效率差，还不方便使用。就是懒得看看 stl 手册。造一些无聊的轮子。

14. 滥用继承机制。从不组合。一说扩展功能就想到继承，甚至多重继承。把类型写得庞大臃肿。最后发现很难不动接口做任何改动。然后完蛋。

15. 不封装数据成员; 滥用静态/全局成员。代码中飞线如乱麻。高度耦合。一有新需求就傻眼。

16. 滥用虚函数。鼓吹一切皆虚。直到某天在构造函数里调了一下。。。

17. 从不画图。业务逻辑混乱，代码层次模糊，对象生存期说不清楚。

18. 滥用 operator/滥用模板机制/滥用宏。追求语法糖。把简单代码写到编辑器推导不出才满意。

19. 不理解浮点值。最常见的是把经过运算的 float 直接和定值比较。出现问题怪 CPU 不靠谱。

20. 无视 warning。总认为认为能通过编译就大功告成了。然后埋一个如分枝无返回值之类的雷到运行时。

21. 分不清平台 API 与 C/C++ 标准库函数。比如用 MFC 在 win 下写服务器。然后发现在 Linux 上无法部署。

22. 不写或不会写测试。从不用代码测试代码，拿测试人员或用户当小白鼠。或把一段简陋的临时代码插到程序某处运行。觉得能跑就删除了。

23. 过度具象的盲目追求“性能”。设计时言必提性能。茴字 100 种写法全都为性能。高估函数调用开销。高估 new/delete 的开销。使用 dowhile 处理分枝。把代码搞得像狗屎。

24. 只用一种技术/语言/平台。一切没听过的东西都不存在。

25. 常见问题不想了解。比如不理解字符编码与传输格式的区别; 不知道各种调用约定的区别。这种太多了。其实就是花几分钟看看书的事。

26. 自己的代码是金口玉言。代码从不重构。也不许别人动。

27. 注释与代码对不上，命名与作用都不上，变量名与类型对不上。看到如“军衔”(rank) 与“角色等级”(playerLevel) 两个变量相比较就想抽人。

28. 神奇的变量前缀。另外即使没有 IDE 提示，我也认为这东西一点用都没有。缩小变量作用范围才是更好的选择。

作者:大狐狸链接:https://www.zhihu.com/question/26134373/answer/91025385 来源: 知乎著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

Foreword of SICP by Alan Perlis

教育者、将领、饮食专家、心理学家，以及父母们，皆编程（规划）。士兵、学生和某些社群，皆被编程（规划）。攻克大型问题，要使用一系列程序，其中大部分产生于求解问题的途中。程序充斥着各种状况，于着手的问题，状况各异。编程作为一项智力活动，为了自个儿能领会，你必须开始使用计算机去编程; 你必须阅读、书写大量的计算机程序。程序是怎样的，又是干什么的，这些都不大重要。重要的是，执行有多快，在构造更大规模的程序过程中，配合其他程序有多流畅。程序员必须努力寻求部分的完善性以及整体的融洽性。本书中，”程序设计 (program)”一词的使用将集中于程序的创建、执行和研究，这些程序用 Lisp 写就，运行于数字式计算机。使用 Lisp，不会限制我们的编程范围，这仅是程序的一种描述记号罢了。

纵观本书主旨，我们将涉及三大关键点: 人的思维方式，计算机程序集，以及计算机本身。程序皆模型，它诞生于人的心智，是真实世界或精神世界的进程。此些进程，起于人的经验和思考，数目巨大，细节复杂，无论何时，都仅能被部分理解。进程通过计算机程序进行模拟，并不能一劳永逸。即便程序是精雕细琢的离散符号集，是交织嵌套的函数，它们依然需要不断演化: 当对模型的理解深化了，扩大了，推广了，我们就修改模型，直到取得相对稳定的状态。接着，又得探寻另一个更好的模型。用计算机编程的欢欣之源，在于思维的不断拓展，在于计算机装置的程序表示，在于其激发的理解力之飞升。艺术诠释梦想，计算机化用程序，实现梦想!

尽管计算机功能强大，它却是个苛刻的工头。它要求程序必须正确，我们的表述必须精细无误。正如其他使用符号的场合，通过论证，我们也将相信程序的正确性。Lisp 本身就能被赋予一套语义 (也就是另一个模型)，如果将程序的函数加以规范化，那么就能用逻辑学的证明方法——谓词演算，来做经得起考验的论证。遗憾的是，一直以来，程序都在变大、变复杂，规范本身的适用性、兼容性，以及正确性也变得值得怀疑。因此，正确性的完全形式化论证很少出现在大型程序中。由于大型程序源于小程序，那么，建立正确性业已确定的、标准化的程序结构的仓库就显得尤其重要了。我们称其为惯用语。接着学习将其组合成更大型的结构，而这就要利用到有价值的组织技术。这些技术都充分展现在本书中，理解它们，对于投入编程这项创造性事业来说，是不可或缺的。最为重要的是，发现并掌握强有力的组织技术，

以增进我们创建大型、重要之程序的能力。反过来说，正是因写大型程序太过费力，才刺激我们去发明新方法，来减少大型程序中的大量函数和细节。

与程序不同，计算机必须遵守物理定律。如果计算机想快速执行——几个纳秒一次状态转换，那么传输的电子就必须限制在小距离内 (最多 1.5 英尺)。热量因大量元器件而产生并汇聚，这必须消除。精湛的工程技艺的开发，平衡了功能多样性与器件密集性间的矛盾。不论何种情况，硬件都比我们所关注的软件工作在更底层。将 Lisp 程序转换为“机器”程序的过程，本身就是用程序所设计的抽象模型。对此过程的研究和创建，为设计其他模型的相关组织技术提供了大量真知灼见。当然，计算机本身亦能被如此建模。试想: 最小的物理开关元件用量子力学建模，量子力学可用微分方程组来描述，微分方程的的细节特性能用数值逼近获得，数值逼近能用计算机上执行的程序来表示，计算机程序由……组成，……!

区分上述三大关键点，不仅是为了方法上的便捷。虽然有人说它们皆存于自然人之头脑，但逻辑上的划分导致三者间符号流动的加速。这使得它们的丰富性、鲜活性，以及无限可能性远胜于其在人类实践中的自然演化（Even though, as they say, it's all in the head, this logical separation induces an acceleration of symbolic traffic between these foci whose richness, vitality, and potential is exceeded in human experience only by the evolution of life itself.）。在最好情况下，三者间的关系是相对稳定的。计算机内存永远不够大，计算速度永远不够快。每一次硬件技术的突破都产生更大规模的程序设计产业，产生新的组织技术，产生更丰富的抽象模型。每个读者都应定期反问自己，“到哪儿才是头，到哪才是头?”——但不要问得太频繁，不然这会让你泥淖在悲喜交加之中，从而丧失编程的乐趣。

我们所写的程序中，有些 (但不够多) 具有精确的数学的功能。比如: 数列排序，找出数列中的的最大值，素数检验，或者计算平方根。我们称如上述的程序为算法，其中大量的因其具有最优性能而被大家所熟知，这里要特别提及两个重要性能参数，程序的执行时间和数据存储要求。程序员应该追求良好的算法和惯用法。虽然有些程序难以精确描述，但程序员有责任去估计并不断地设法改进程序的性能。

Lisp 是个幸存者，已被使用了约四分之一世纪。在用的语言中，只有 Fortran 比它久远。这两语言都支持某些重要的应用领域的程序设计需要。Fortran 用于科学和工程计算，Lisp 用于人工智能。这两领域将一如既往地重要，其中程序员是如此倾心于 Lisp 和 Fortran，以致它们可能继续被活跃使用至少四分之一世纪。

Lisp 一直在改变着。本书使用的方言 Scheme 就是演化自最初的 Lisp，并与后者在若干重要方面有所不同，包括变量约束的静态作用域，以及允许函数生成函数作为函数值。在语义结构方面，与早期 Lisp 族相比，Scheme 更加接近于 Algol 60。Algol 60 再也不能重新流行了，只能存活在 Scheme 和 Pascal 的基因里。比起萦绕其周围的其他语言，很难找到像这样两种语言去更好地沟通两个差异如此之巨大的文化。（It would be difficult to find two

languages that are the communicating coin of two more different cultures than those gathered around these two languages.）Pascal 是用来建造金字塔的——由大群人推笨重的大石到指定地点，壮丽辉煌、令人震撼、静态的结构。Lisp 是用来建造有机体的——由小分队把无数更简单的处于变化之中的有机体安置在指定位置，壮丽辉煌、令人震撼、动态的结构。组织原理在两语言中是相同的，除了极其重要的一点不同外: 把自由输出功能托付给 Lisp 程序员个体。这在 Pascal 程序员那里也能找到，却降低了不止一个数量级。Lisp 程序大大充实了函数库，这些函数的实用性超过了催生它们的应用。表，Lisp 的固有数据结构，为函数实用性的提升贡献巨大。表的简单结构和天然的适用性反映在函数，就是函数那令人惊讶的普适性。而在 Pascal，数据结构的过度声明导致函数的专用性，这阻碍并恶化了函数间的临时配合。拥有 100 个函数，工作在 1 个数据结构之上，好过拥有 10 个函数，工作在 10 种数据结构之上。结果，金字塔必定是矗立千年而不变，有机体要么演化，要么死亡。

为了看清上述差异，请试着比较 Pascal 入门书与本书对材料和练习的处理方法。不要困于假象，说，这是本仅适用于 MIT 的教科书，它的独特性只不过是因为它出自 MIT. 准确地说，任何一本 Lisp 编程书籍都应该如本书，无论其学生是谁，书本在何处使用。

注意，本书是关于程序设计的，异于大多数 Lisp 书籍，因那些书是为人工智能准备的。总之，随着被研究的系统的不断扩大，程序设计的关键问题，在软件工程和人工智能之间将趋于统一。这就解释了为何在人工智能领域之外，对 Lisp 的兴趣也正不断地在提高。

从人工智能的目标出发，可以预见，其研究将产生大量有意义的程序设计问题。在其他程序设计文化里，这一连串的问题将引生新的语言。诚然，在任何极大型的程序设计工作中，一个有效的组织原理就是控制并隔离任务 (作业) 模块间的信息流动，而这要通过新语言的发明方可实现。当人们接近到与之最常交互的系统的边界时，这些语言趋于变得高级。结果，这些系统就包含了大量重复的复杂的语言处理功能。Lisp 拥有简洁的语法和语义，这使得语法分析被看作是一项基本任务。这使得语法分析技术在 Lisp 程序里几乎没有地位，同时使得语言处理器的构造对大型 Lisp 系统的发展变更速度而言，鲜有阻碍。最后，正是语法和语义的极端简洁性，导致了 Lisp 程序员的自由以及负担。任何规模的 Lisp 程序，除了寥寥数行的之外，无不充盈着各类函数。发明函数，组装函数; 得到函数，再利用之发明函数（Invent and fit; have fits and reinvent!）！让我们举杯庆贺那些将思想写进层层括号之巢的 Lisp 程序员吧。

Alan J. Perlis

纽黑文市, 康涅狄格

作者:拉曼卡链接:https://www.zhihu.com/question/20246883/answer/45506752 来源：知乎著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

3.18 efficient c++ notes

efficient c++ notes

- 1 什么 RAII、异常已经有人说了，我再补充一些。。。（顺便反对某些回答不懂装懂，什么面向对象啊到底写没写过 C++ = =）

    1. 在写一个 raw loop 之前，一定要考虑：我要写的东西 <algorithm> 能不能搞定？一定要善于利用 <algorithm>，一定！（而且利用 <algorithm> 可以秒杀各种笔试、面试题，部分可参考陈硕大大的文章）而且要意识到 iterator 与 algorithm 组合的强大威力，例如倒序打印字符串：

    copy(crbegin(str), crend(str), std::ostream <char>{std::cout});

    1. 在可能的情况下，要尽量使用非成员版函数（当年 STL 之父本来就是要把 begin() end() 这些做成非成员函数的，就是怕和标准委员会打起来才做成了成员函数。当然，现在开始陆陆续续擦屁股了。。。）例如

    using std::begin; using std::end; sort(begin(vec), end(vec)); 当然，对于 std::list/forward 这种特殊的数据结构可能需要使用成员函数（例如 sort、reverse）以效率最大化。

    1. 意识到 C++ 的“多态”远远不止所谓的虚函数（Tutorial - 1.60.0【伪处贴】关于 C++14 后 boost::variant 变得异常好用の件）

    作者：刘雨培链接：https://www.zhihu.com/question/41424757/answer/91037477 来源：知乎著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

- 2 不光是不能学成“C 风格的 C++”，更重要的是不能学成“Java 风格的 C++”，前者还可以抢救一下，后者就只能放弃治疗了……0、不要用裸指针，实在忍不住就用“世界上最傻的 smart pointer”1、不要用 malloc/free，也不要用 new/delete，用 smart pointer 管理对象的所有权和生存期 2、不要用函数指针，也不要用 pure interface 3、不要用原生数组，用 STL 里的容器

    作者:徐辰链接:https://www.zhihu.com/question/41424757/answer/91048974 来源：知乎著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

3.18 需要更多的“好的表达”

需要更多的“好的表达”

今晚灯下仔细反思今年的关键词里有什么没做的好的，对于“好的习惯，好的思考”都还满意，而“好的表达，好的交流”实践的太少。那我今后就要努力地贯彻这两点。

TODO Aim to study ESL & Data Mining notes & HWs from 3.21 to end of April

This is an important step toward future interview. Should start early!

TODO 翻译 project

简单明快

3.19 ” 乐” 的境界

” 乐” 的境界

1. 学而时习之，不亦乐乎
2. 知之者不如好之者，好之者不如乐之者

编程语言的” 语感”(坑)

” 志，气，力” 的三位一体与”if.... then ....” 的思考范式

It is better to learn C++(and any complicated thing) by concrete examples/projects

If you want to finish reading all the rules first then start writing code, it is going to be late and unnecessary.

3.22 忠实的、快乐的自我表达

忠实的、快乐的自我表达

今天上完班早早的回到家就拿起了篮球一个人在小区球场练球。练着练着，就发现手感这些的找回和单纯快乐地打球的关系太大了，不是说不该有争胜心，而是哪怕的争胜的时候也应该去享受打球那种纯粹的快乐。推而广之，在做任何事情之前，或许都可以问问自己，这件事自己能快乐开心地去做吗？如果可以，怎么去从中发掘最大的快乐和有趣？

编程和算法有趣且让我快乐，所以我一定会在这条路上走下去，不断提升自己。数学的很多分支和问题对我来说也是如此。统计呢？统计的不少东西让目前的我觉得 tedius（比如在公司读的不少 paper），但概率论在博弈等方面的应用我有兴趣，再比如在公司接触的因果推断我也觉得蛮有意思的。

至于机器学习，统计学习，不得不说我虽然觉得很有用，但我在这方面积累的让我“开眼界”或者说“震撼我”的例子太少了。说到底还是太浅尝辄止了。前两天下了决心这两月好好整一整这块的内容，就要从实际的例子出发，相信通过努力一定能见识到让我“开眼界”的东西！

3.25 Principle of Inclusion & Exclusion(really powerful)

PIE

- Probability form
- Waring formula
- He Min's master qualifying problems

Kenneth Lange's Applied Probability book is dense and fun

I plan to buy the book from the following link: `http://www.barnesandnoble.com/p/applied-probability-kenneth-lange/1119397767/2672928983758`

3.31 代码的反编译

代码的反编译

要理解这个问题，先要看「正」编译的过程是怎样的。你有一个想法，这是一种人类自然语言可以表达出来的东西。你利用编程技能，把它「翻译」成你熟悉的一种编程语言:

然后编译器（compiler）将它翻译成机器所能理解的语言:

这中间的每一步都是一个「信息丢失」的过程。比如你说，我要把这些数排个序。然后你轻车熟路地写了个冒泡排序。你的原始动机其实已经一定程度上从你的代码里丢失了——有经验的人可能可以一眼看出你这段代码是在排序，而新手小明看到的只有一些 for 和 if 之类的东西。如果是更复杂的功能，那么可能过一段时间你自己都看不懂自己当时是想干什么。从程序语言到机器语言的过程其实也是一样的。这两个过程其实都是把「做什么」转换成「怎么做」的过程，转换完成之后，究竟一开始是要做什么，这个信息已经丢失了。

所谓「反编译」，其实就是找回这些丢失的信息的过程。从这个角度上来说，你阅读一段代码的过程，其实就是在将它「反编译」成自然语言。如果

要完美地反编译，那只存在一种可能，就是信息完全没有丢失——比如说你阅读的这段代码有充分的注释，或者它使用了一种你所知晓的模式（这也是为什么大家一再强调注释和设计模式的重要性）。对于从机器语言到程序语言的反编译过程，也是一样。

作者:hillin 链接:https://www.zhihu.com/question/21853681/answer/74134768 来源：知乎著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

如何在 VS 中正确输出中文，或者说使其支持 Unicode?

- code example(check the storage allocation in bytes for most primitive types) // to see the storage allocation in bytes for most primitive types //#include<iostream> //#include<string> //#include <limits> */using namespace std; /*

- comparison & comments 以上 code，在 VS 2015 下编译打印出来的中文皆乱码，而在 spacemacs 下编译打印则完全正确显示。

April

4.1 Use Mathematica to produce function plots

Mathematica plotting eg

Plan to get more familiar with Latex math writing by recording down notes via Latex

4.3 Tax Return filed

Tax Return information

- state tax Confirmation Number: 1707036

4.4 一六年健身和饮食计划 body

肚子胀，需要开始坚持跑步还有更规律地饮食

原因未知，不知道和四五天前吃了二十颗 multivitamin 有关。如果通过跑步健身也消不下去就得去看医生。还有就是，减少坐的时间，一天即使不跑步也尽量不要久坐，散散步也好。

一六年健身和饮食计划

1）把肚皮瘦下来。从四月到十月，每周至少跑十公里，有时间的话尽量每隔一天早上跑

2）想想怎样练腹肌。目前的想法是每天至少三十个俯卧撑。

3）加强上肢力量，想想有没有法子搭一个可以做引体向上的器械。

4）阅读 Bruce Lee:Fighting Technique 一书

5）多喝水多吃蔬菜（比如菠菜多吃），少吃脂肪和胆固醇含量高的食物，少喝饮料（偶尔想调节口味的时候可以喝少量），适当喝牛奶。

6）形成吃早餐的习惯。中午和晚上控制饭量，不超过半碗（一两）。

7）避免久坐。养成每坐一两小时起来走动走动压压腿的习惯以应对无法避免之状况。

8）每天都要适当进行在站立或走动的时候思考，在家可以考虑在白板上做题以及在用电脑的时候把显示器和键盘垫高来操作。

4.8 究竟什么使你感动?! & 公平的时间

究竟什么使你感动?! What is your motivation?!

如果真的觉悟到了这点，那“奔”一字足矣。

公平的时间

不管你一天做什么，请在每天开始的时候都想到————时间对每个人是公平的，你现在花时间做的事所产生的影响或许终有一天会积土成山。不要荒废每一天，不要三天打鱼两天晒网。

许宝騄

他在教学中特别强调“直观地理解数学”的重要性，他主张要把数学定理及其证明的“原始思想”告诉学生，总是殷切地期望学生能直观地领悟数学命题的来龙去脉。在分析问题时，他强调要有一种“内视”能力，认为“数学中的抽象能力很重要，一些问题经过抽象后，不仅简明而且其实质也清楚了”。徐利治曾说，他从许宝騄身上学到了“不怕计算”和“乐于计算”的习惯，十分乐于从计算中发现规律和提炼一般性公式。Biuman 写道：“许坚持深入浅出，毫不回避困难，特别是深沉、明确而默默投身于学术的最高目标和水准，这些精神吸引了我们。”许看书有个习惯，就是把书中每一道题都认真做一遍。他说：“若这道题很简单那就不费时间；若不易做，那正说明你应该做它。”

丘成桐

1. 面对一本很难懂的数学书，我总是锲而不舍地反复阅读，从中获益匪浅。

2. 我逐渐地能有效地消化所学内容了。更觉欣喜的是，当后来需要运用某些概念去理解新问题时，以往所学常蓦然浮现，令人茅塞顿开。

3. 长期沉浸于自己钟爱的领域，虽然未必能取得立竿见影的回报，但那种润物无声的渗透与潜移默化的熏陶，无时不在影响着科学研究。

4.15 Never forget your motivation!

Never forget your motivation!

4.16 跟波利亚学解题（刘未鹏）

跟波利亚学解题（刘未鹏）

- 一些方法 总结波利亚在书中提到的思维方法，尤其是《How To Solve It》中的启发式思考方法，有这样一些：

    1. 时刻不忘未知量（即时刻别忘记你到底想要求什么，问题是什么。）莱布尼兹曾经将人的解题思考过程比喻成晃筛子，把脑袋里面的东西都给抖落出来，然后正在搜索的注意力会抓住一切细微的、与问题有关的东西。事实上，要做到能够令注意力抓住这些有关的东西，就必须时刻将问题放在注意力层面，否则即使关键的东西抖落出来了也可能没注意到。
    2. 用特例启发思考。一个泛化的问题往往给人一种无法把握、无从下手、或无法抓住里面任何东西的感觉，因为条件太泛，所以看起来哪个条件都没法入手。一个泛化的问题往往有一种“不确定性”（譬如元素的个数不确定，某个变量不确定等等），这种不确定性会成为思维的障碍，通过考虑一个合适的特例，我们不仅使得问题的条件确定下来从而便于通过试错这样的手法去助探问题的内部结构，同时很有可能我们的特例中实质上隐藏了一般性问题的本质结构，于是我们便能够通过对特例的考察寻找一般问题的解。

    1. 反过来推导。反过来推导是一种极其重要的启发法，正如前面提到的，Pappus 在他的宏篇巨著中将这种手法总结为解题的最重要手法。实际上，反向解题隐含了解题中至为深刻的思想：归约。归约是一种极为重要的手法，一个著名的关于归约的笑话这样说：有一位数学家失业了，去当消防员。经过了一些培训之后，正式

上任之前，训练的人考他：如果房子失火了怎么办？数学家答出了所有的正确步骤。训练人又问他：如果房子没失火呢？数学家答：那我就把房子点燃，这样我就把它归约为了一个已知问题。人类思维本质上善于“顺着”推导，从一组条件出发，运用必然的逻辑关系，得出推论。然而，如果要求的未知量与已知量看上去相隔甚远，这个时候顺着推实际上就是运用另一个启发式方法——试错——了。虽然试错是最常用，又是也是最有效的启发法，然而试错却并不是最高效的。对于许多题目而言，其要求的结论本身就隐藏了推论，不管这个推论是充分的还是必要的，都很可能对解题有帮助。如果从结论能够推导出一个充要推论，那么实际上我们就将问题进行了一次“双向”归约，如果原问题不容易解决，那么归约后的问题也许就容易解决了，通过一层层的归约，让逻辑的枝蔓从结论上一节节的生长，我们往往会发现，离已知量越来越近。此外，即便是从结论推导出的必要非充分推论（“单向”归约），对问题也是有帮助的——任何不满足这个推论的方案都不是问题的解：譬如通过驻点来求函数的最值，我们通过考察函数的最值（除了函数边界点外），发现它必然有一个性质，即在这个点上函数的一阶导数为 0，虽然一阶导数为 0 的点未必是最值点，但我们可以肯定的是，任何一阶导数不为 0 的点都可以排除，这就将解空间缩小到了有穷多个点，剩下的只要做做简单的排除法，答案就出现了。再譬如线性规划中经典的单纯形算法（又见《Algorithms》），也是通过对结论的考察揭示出只需遍历有限个顶点便必然可以到达最值的。此外很多我们熟知的经典题目也都是这种思路的典范，譬如《How To Solve It》上面举的例子：通过一个 9 升水的桶和一个 4 升水的桶在河里取 6 升水。这个题目通过正向试错，很快也能发现答案，然而通过反向归约，则能够不偏不倚的命中答案。另一些我们耳熟能详的题目也是如此，譬如：100 根火柴，两个人轮流取，每个人每次只能取 1~7 根，谁拿到最后一根火柴谁赢；问有必胜策略吗，有的话是先手还是后手必胜？这个问题通过试错就不是那么容易发现答案了。同样，这个问题的推广被收录在《编程之美》里面：两堆橘子，各为 m 和 n 个，两人轮流拿，拿的时候你只能选择某一堆在里面拿（即不能跨堆拿），你可以拿 1~ 这堆里面所有剩下的个橘子，谁拿到最后一个橘子谁赢；问题同上。算法上面很多聪明的算法也都是通过考察所求结论隐藏的性质来减小复杂度的，譬如刚才提到的单纯形问题，譬如经典面试题“名人问题”、“和最小（大）的连续子序列”等等。倒推法之所以是一种极为深刻的思维方法，本质上是因为它充分利用了题目中一个最不易被觉察到的信息——结论。结论往往蕴含着丰富的条件，譬如对什么样的解才是满足题意的解的约束。一般来说，借助结论中蕴含的知识，我们便可

以更为“智能地”搜索解空间。举一个直白的例子，有人要你在地球上寻找一栋束条件）。对于这样一个问题，最平凡的解法是穷举地球上每一栋建筑，直到遇到一个满足条件的为止。而更“智能”的（或者说更“启发”的）方法则是充分利用题目里面的约束信息，譬如假若条件里面说要 60 层楼房，你就不会去非洲找，如果要拜占庭风格的，你估计也不会到中国来找，如果要始建于很早的年代的，你也不会去非常新建的城市里面去找，等等。倒推法是如此的重要，以至于笛卡尔当时认为可以把一切问题归结为求解代数方程组，笛卡尔的万能解题法就是首先将问题转化为代数问题，然后设出未知数，列出方程，最后解这组（个）方程。其中设未知数本质上就是一种倒推：通过设出一个假想的结论 x，来将题目对 x 的需求表达出来，然后顺势而下推导出 x。仔细想想设未知数这种手法所蕴含的深刻思想，也就难怪笛卡尔会认为它是那个解决所有问题的一般性钥匙了。

1. 试错。试错估计是世界上被运用最广泛的启发法，你拿到一个题目，里面有一些条件，你需要求解一个未知量。于是你对题目这里捅捅那里捣捣，你用上所有的已知量，或使用所有你想到的操作手法，尝试着看看能不能得到有用的结论，能不能离答案近一步。事实上，如果一个问题的状态空间是有限的话，往往可以通过穷举所有可能性来找到那个关键的性质。譬如这样一个问题：有一个囚犯，国王打算处决他，但仁慈的国王给了他一个生还的机会。现在摆在他面前有两个瓶子，一个里面装了 50 个白球，一个装了 50 个黑球，这个囚犯有一个机会可以随便怎样重新分配这些球到两个瓶子中（当然，要保证不空），分配完了之后囚犯被蒙上眼睛，国王随机取一个瓶子给他，他在里面摸出一个球（因为蒙着眼睛，所以也是随机抽取），如果白球，则活，否则挂掉。问，这个囚犯如何分配，才能最大化生还几率。结合特例和试错法，这个题目的答案是很容易发现的。这样的题目还有很多。实际上，历史上很多有名的发现也都是无意间发现的（可以看作是试错的一种）。

1. 调整题目的条件（如，删除、增加、改变条件）。有时候，通过调整题目的条件，我们往往迅速能够发现条件和结论之间是如何联系的。通过扭曲问题的内部结构，我们能发现原本结构里面重要的东西。譬如这样一个题目（感谢 alai 同学提供）：A 国由 1000000 个岛组成，岛与岛之间只能用船作为交通工具，有些岛之间有船来往，从任意一个岛都可以去到另外任一个岛，当然其中可能要换船。现在有一个警察要追捕一个逃犯，开始时他们在不同的岛上，警察和逃犯都是每天最多乘一次船，但这个逃犯还有点迷信，

每个月的 13 日不乘船，警察则不迷信。警察每天乘船前都知道逃犯昨天在哪个岛上，但不知道他今天会去哪个岛。请证明，警察一定可以抓到逃犯（即到达同一个岛）。通过拿掉题目中一个关键的条件，观察区别，然后再放上那个条件，我们就能“感觉”到题目的内在结构上的某种约束，进而得到答案。

1. 求解一个类似的题目。类似的题目也许有类似的结构，类似的性质，类似的解方案。通过考察或回忆一个类似的题目是如何解决的，也许就能够借用一些重要的点子。然而如何在大脑中提取出真正类似的题目是一个问题。所谓真正类似的题目，是指那些抽象结构一样的题目。很多问题表面看是类似的，然而抽象结构却不是类似的；另一些题目表面看根本不像，然而抽象层面却是一致的。表面一致抽象不一致会导致错误的、无效的类比；而表面不一致（抽象一致）则会阻碍真正有用的类比。《Psychology of Problem Solving》里面对此有详细的介绍。后面也会提到，为了便于脑中的知识结构真正能够“迁移”，在记忆掌握和分析问题的时候都应该尽量抽象的去看待，这样才能够建立知识的本质联系，才能够最大化联想空间。

1. 列出所有可能跟问题有关的定理或性质。这个不用说，我们在最初学习解题的时候就是这么做的了。

1. 考察反面，考察其他所有情况。很多时候，我们在解题时容易陷入一种特定的手法，比如为什么一定要是构造式的来解这个题目呢？为什么不能是逼近式的？为什么一定要一步到位算出答案？为什么不能从一个错误的答案调整到正确答案？为什么这个东西一定成立？不成立又如何？等等。经典例子：100 个人比赛，要决出冠军至少需要赛多少场。
2. 将问题泛化，并求解这个泛化后的问题。刚才不是说过，应该通过特例启发思考吗？为什么现在又反倒要泛化呢？实际上，有少数题目，泛化之后更容易解决。即，解决一类问题，比解决这类问题里面某个特定的问题还要容易。波利亚称之为“发明者悖论”，关于“发明者悖论”，《数学与猜想》第一卷的开头有一个绝妙的例子，可惜这里空间太小，我就不摘抄了 - _-|||
3. 下意识孵化法。这个方法有点像老母鸡孵小鸡的过程：我们先把问题的吃透，放在脑子里，然后等着我们的下意识把它解出来。不过，不宜将这个方法的条件拉伸过远，实际上，除非能够一直保持一种思索的状态（金出武雄所谓“思维体力”），或者问题很简单，否则一转头去做别的事情之后，你的下意识很容易就把问题丢开了。据说庞加莱有一次在街上，踏上一辆马车的那一瞬间，

想出了一个重要问题的解。其他人也像仿效，结果没一个人成功。实际上，非但马车与问题无关，更重要的是，庞加莱实际上在做任何事的时候除了投入有限的注意力之外，其他思维空间都让给了那个问题了。同样，阿基米德从浴缸里面跳出来也是如此；如若不是经过了极其痛苦和长时间的思索，也不会如此兴奋。如果你也曾经花过几天的时间思考一个问题，肯定也是会有类似的经历的。

1. 烫手山芋法。说白了，就是把问题扔给别人解决。事实上，在这个网络时代，这个方法有着无可比拟的优越性。几乎任何知识性的问题，都可以迅速搜索或请教到答案。不过，如何在已知知识之外发掘出未知知识，如何解决未知问题，那就还是要看个人的能力了。数学界流传一个与此有关的笑话：如果你有一个未解决问题，你有两个办法，一，自己解决它。二，让陶哲轩对它感兴趣。

- 联想的法则 人类的大脑是一个复杂而精妙的器官，然而某种程度上，人类的大脑也是一个愚蠢的器官。如果你总结过你解过的一些有意义的好题目，你会发现它们有一个共同点：没有用到你不知道的知识，然而那个最关键的、攸关成败的知识点你就是想不到。所以你不禁要问，为什么明明这个知识在我脑子里（也就是说，明明我是“能够”解决这个问题的），但我就是没法想到它呢？“你是怎么想到的?”这是问题解决者最常问的一个问题。甚至对于熟练的解题者来说，这个问题的答案也并不总是很明确的，很可能他们自己也不清楚那个关键的想法是怎么“蹦”出来的。我们在思考一个问题的时候，自己能意识到的思维部分似乎是很少的，绝大多数时候我们能感知到的就是一个一个的转折点在意识层面显现，我们的意识就像一条不连续的线，在其上的每一段之间那个空档内发生了什么我们一无所知，往往我们发现被卡在一个地方，我们苦思冥想，然后一个知识（也许是一个性质，也许是一个定理）从脑子里冒了出来，或者说，被我们意识到，然后我们沿着这条路走一段，然后又卡住，然后又等待一个新的关键知识的出现。而至于这些知识是怎么冒出来的？我们可以对它们的“冒出来”提供怎样的帮助？我们可以在意识层面做一些工作，帮助我们的下意识联想到更多重要的知识吗？那些灵光一现的瞬间，难道只能等待它们的出现？难道我们不能通过一些系统化的步骤去“捕获”或“生成”它们？又或者我们能不能至少做些什么工作以使得它们更容易发生呢？

  正如金出武雄在《像外行一样思考，像专家一样实践》中所说的，人类的灵感一定是有规律的，认知科学目前至少已经确认了人类思维的整个物质基础——神经元。而既然它们是物质，自然要遵循物质的运行规律。只不过我们目前还没有窥破它们，但至少我们可以确信的是，它们在那里。事实上，不需要借助于认知科学，单单是通过对我们自己

思维过程的自我观察，也许就已经能够总结出一些重要的规律了，也许，对自身思维过程的反观真的是人有别于其它动物的本质区别。

《专注力》当中有这样一个例子：一天夜里，你被外面的吵闹声叫醒了，你出去一看，发现有一群人，其中有一个人开着很名贵的轿车，他跟你说他们正在玩一个叫“拾荒者”的游戏，由于一些原因，他必须要赢这个游戏，现在他需要一块 1.5m*1m 的木板，如果你能帮忙的话，愿以一万美元酬报。你怎么办？被测试的大多数人都没有想到，只要把门拆给他就可以了（如果你想到了，祝贺你:-)），也许你会说现在的门都是钢的，没关系，那你有没有想到床板、立柜的门、大桌子的桌面之类的？这个问题测试的就是心理学上所谓的“范畴陷阱”，“木板”这个名词在你脑子里的概念中如果是指“那些没有加工的，也许放在木材厂门口的，作为原材料的木板”的话，那么“木板”就会迅速在你的下意识里面建立起一个搜索范畴，你也会迅速的反应到“这深更半夜叫我上哪去找木板呢?”如果你一下就想到了，那么很大的可能性是“木板”这个概念在你脑子里的范畴更大，更抽象，也许包含了所有“木质的、板状的东西”。

这就是联想的法则。

我们的大脑无时无刻不在对事物进行归类，实际上，不仅是事物，一切知识，都在被自动的归类。在有关对世界的认知方面，被称为认知图式，我们根据既有的知识结构来理解这个世界，会带来很大的优势。实际上，模块化是一个重要的降低复杂性的手段。然而，知识是一把双刃剑，一方面，它们提供给了我们解决问题的无以伦比的捷径优势，“砖头是砌墙的”，于是我们遇到砌墙这个问题的时候就可以迅速利用砖头。然而另一方面，知识却也是思维的桎梏。思维定势就是指下意识遵循既有知识框架思考的过程。上面的那个木板的例子也是思维定势的例子。每一个知识都是一个优势，同时又是一个束缚。著名的科幻作家阿瑟·克拉克有一句名言：如果一位德高望重的老科学家说某个事情是不可能的，那么他很可能是错的。所以，如何在获取知识优势的同时，防止被知识束缚住，是一门技术。

掌握这门技术的钥匙，就是抽象。在吸收知识的时候进行抽象，同时在面对需要用到知识的新问题时也要对问题进行抽象。就以大家都知道的“砖头”有多少种用途为例，据说这道题目是用于测试人的发散思维的，能联想到的用途越多，思维定势就越小。实际上，借助于抽象这个利器，这类题目（乃至更广的一类问题）是可以系统性的进行求解的，我们只需对砖头从各个属性维度进行抽象。譬如，砖头是——长方形的（长方形的东西有什么用途？还有哪些东西也是长方形的，它们都有什么用途?）、有棱角的（问题同上）、坚硬的、固体、有一定大小的体积的、红色的、边界线条平直的、有一定重量的…对于每一个

抽象，我们不妨联想还有其他什么物体也是具有同样抽象性质的，它们具有同样的用途吗？当然，除了抽象之外，还有“修改”，我们可以在各个维度上对砖头的属性进行调整，以期得到新的属性：譬如大小可以调整、固体可以调整为碎末、棱角可以打磨、重量也可以调整、形状也可以调整…然后看看新的属性可以如何联想开去。

除了这个简单的例子之外，我们也不妨看一看一些算法上的例子，同样一个算法，不同的人来理解，也许你脑子里记得的是某个特定的巧妙技巧（也许这个技巧在题目的某步关键的地方出现，从而带来了最令人意外的转折点），然而另一人个记得得也许是“递归”这种手法，还有另外一个人记得的也许是“分治”这种更一般化的解题思路。从不同的抽象层面去掌握这道题目的知识信息，以后遇到类似的问题，你能够想起这道题所提供的知识的可能性是有极大的差异的。《Psychology of Problem Solving》的第 11 章举了这样一个例子：先让被试（皆为大学生）阅读一段军事材料，这个材料是说一小撮军队如何通过同时从几个不同方向小规模攻击来击溃一个防守严实的军事堡垒的。事实上这个例子的本质是对一个点的同时的弱攻击能够集聚成强大的力量。然后被试被要求解决一个问题：一个医生想要用 X 射线杀死一个恶性肿瘤，这个肿瘤只可以通过高强度的 X 射线杀死，然而那样的话就会伤及周围的良好组织。医生应该怎么办呢？在没有给出先前的军队的例子的被试中只有 10% 想到答案，这是控制基线。然后，在先前学习了军队例子的被试中，这个比例也仅仅只增加到 30%，也就是说只有额外 20% 的人“自动”地将知识进行了转移。最后一组是在提醒之下做的，达到了 75%，即比“自动”转移组增加了 45% 之多。这个例子说明，知识的表象细节会迷惑我们的眼睛，阻碍我们对知识的运用，在这个例子中是阻碍问题之间的类比。

而抽象，则正是对非本质细节去枝减叶的过程，抽象是我们在掌握知识和解决问题时候的一把有力的奥卡姆剃刀。所以，无论是在解题还是在学习的过程中，问自己一个问题“我是不是已经掌握了这个知识最深刻最本质的东西”是非常有益的。

- 知识，知识 如果你是一个熟练的解题者，你也许会发现，除了一些非常一般性的、本质的思维法则之外，将不同“能力”的解题者区分开来的，实际上还是知识。知识是解题过程中的罗塞塔碑石。一道几何题为什么欧几里德能够做出来我们不能，是因为欧几里德比我们所有人都更了解几何图形有哪些性质，借助于一个性质，他很容易就能抵达问题的彼岸；反之，对于不知道某个性质的我们，倒过来试图“发现”需要这样的性质有时几乎是不可能的。有人说数学是在黑暗中摸索的学科，是有道理的。并不是所有的问题都能够通过演绎、归纳、类比等手法解出来的。这方面，费马大定理就是一个绝好的例子，《费马大

定理：一个困惑了世间智者 358 年的谜》一书描述了费马大定理从诞生到被解决的整个过程，事实上，通过对费马大定理本身的考察，几乎是毫无希望解决这个问题的，我们根本不能推导出“好，这里我只需要这样一个性质，就可以解决它了”，也许大多数时候我们可以，但那或者是因为我们有已知的知识，或者这样的归约很显然。而对于一些致命的问题，譬如费马大定理，最重要的归约却是由别人在根本不是为了解决费马大定理的过程中得出来的。运气好的话，我们在既有的知识系统中会有这样的定理可以用于归约，运气不好的话，就得去摸索了。

所幸的是，绝大多数问题并不像费马大定理这样难以解决。而且绝大多数问题需要用到的知识，在现有的知识系统里面都是存在的。我们只要掌握得足够好，就有希望联想起来，并用于解题。

当然，也有许多题目，求解它们的那个关键的知识可以通过考察题目本身蕴涵的条件来获得，这类题目就是测试思维本身的能力的好题目了。而如果这个性质根本无法通过对题目本身的考察得出来，那么这个题目测试的就是知识储备以及联想能力。

- 好题目，坏题目 在我看来，好题目即测试一个人思维的习惯的题目（因为知识性的东西是更容易弥补的，尤其是在这样一个年代；而好习惯不是一朝一夕养成的），它应有这样一些性质：

  不需要用到未知的知识，或者需要用到未知的知识，但一个敏锐的解题者可以通过对题目的分析自行发现这些所需的知识。考察解题的一般性思路，而不是特定（ad hoc）的解题技巧，尤其是当这个技巧几乎不可能在短时间内通过演绎和试错发现的时候。譬如题目需要用到某种性质，而这个性质对于不知道它的人来说几乎是无法从对题目的考察中得出来的。考察思维能力：联想能力、类比能力、抽象能力、演绎能力、归纳能力、观察能力、发散能力（思维不落巢臼的能力）。考察一般性的思维方法：通过特例启发思考、通过试错寻找规律、通过泛化试探更一般性命题、通过倒过来推导将问题进行归约、通过调整（分解、删除、增加等等）题目的条件来感知它们之间的联系以及和结论的联系、通过系统化的分类讨论来覆盖每种可能性。好题目举例：烙饼排序问题（考察特例启发法以及观察能力）、Nim 问题（还有简单版本的取火柴问题）（烙饼排序问题和 Nim 问题可参见《编程之美》）、9 公升 4 公升水桶倒 6 公升水的问题（考察倒过来思考问题的能力）、9 点连线问题、6 根火柴搭出 4 个面的问题、“木板”问题（考察思维定势，此外《心理学与生活》的第九章也有好几个经典的问题）、许多数论问题（观察能力、演绎能力、归纳能力）。此外，我们最近也在讨论好题目。而坏题目呢：

  好题目各有各的好，坏题目都是相似的。坏题目基本上就是指那些所

谓的 unfair questions，什么是 unfair，举个例子：一个人住在一栋非常高的楼上，每天早晨他乘电梯下到一楼，出门上班。但晚上回来之后却最多只能坐到一半高度的楼层，剩下一半只能走楼梯上去，除非是下雨天。问为什么。这个例子据说不少人小时候在脑筋急转弯里面做过，但我很怀疑基本上任何正常人是不是可能想出来。这个问题的问题在于他需要用到千百个有可能与问题有关的性质中的一个，而且这个性质还根本无法通过对题目本身的考察得出来，只可能某天我们碰巧遇到类似的场景也许才能想到。知道答案的人也许会说答案很显然，但别忘了心理学上的事后偏见——一旦知道结果之后，所有指向结果的证据看上去都那么显然和充分，而同时所有反结果的证据看起来都那么不显然和不充分。譬如这题关键是要想到这人是矮子和雨天要带伞，也许你会说“只要考虑一下电梯的按钮面板就会发现了”，或者“看到下雨，那还不想到带伞么?”，然而这只是事后的合情推断。在不知道答案的情况下，这个故事中有数不清的因素可能会成为问题的解释，除非某天我们碰到类似的问题，否则大致也只能一个个穷举了去使劲往上凑，譬如除了身高之外还有：是不是瞎子、是不是聋子、是不是哑子、男人女人、什么牌子的电梯、大厦是哪种大厦？这些因素重要吗？不重要吗？最令人头疼的是，在不知道答案的时候，我们也根本不知道他们重不重要，一个出谜语的人可能从任何一个微小的地方引申出某个谜语来；更头疼的是，我们不知道我们不知道的那些因素是不是也可能与题目的解有关，譬如这样一个问题：一个人走进酒吧，问酒保要一杯水，酒保掏出一只枪，拉上扳机；这人说声“谢谢”，走了出去。这些题目固然有趣，但几乎没有价值。值得注意的是，这样的问题跟著名的 9 点连线问题和 6 根火柴搭出 4 个面的问题（还有《如何解题：现代启发式方法》里面那个经典的“小球在盒内碰撞何时回到原轨迹”的问题）不同，后者的条件都在眼前，并且解的搜索空间无论如何很小，就看思维能不能突破某一个框框。而上面这些问题则是要人进行根本不可能的联想。9 点问题实际上是可以系统化思考解决的，但 unfair question 则像许多谜语一样，随便哪个人都可以出一个另一个人根本无法想出来的谜语，因为从谜语隐含的信息加上人可能从谜语中联想出来的信息，加起来也不足以构成解题的充分条件；这种情况下除非你遇到出题人在出题时的心理或所处情况，否则是无法解的。最后，发散性思维其实是可以系统化的，参见前文“联想的规则”。出题的误区：

最大的误区就是把知识性的题目误当成能力型的题目。如果题目中需要用到某个重要的定理或性质，而对于一个原本不知道这个定理或性质的人来说是无法通过题目本身到达这个性质的，那这就属于知识性的题目。虽然几乎所有题目归根到底都是知识性的，但有些题目更为知识性，尤其是当解题中需要用到的定理或性质并不那么 trivial 的时

候。一个最好的题目就是问题明明白白，而且最终的解也没有用到什么神秘的定理，但要想获知到解，取决于你会不会思考一个问题（参见“好问题”）。譬如烙饼问题和 Nim 问题，还有许许多多问题简洁明确但很锻炼思考的算法问题。

- 一个好习惯 在解题的过程中，除了必要条件——知识储备——之外，对于一些并不涉及什么你不知道的定理的题，很大程度上就要看思维能力或者习惯了。而在思考一个问题的时候，最容易犯的一类错误就是忘了考虑某种可能性，不管这种可能性是另一种做法（譬如只顾着构造一个能一步得出结果的算法，没记得还可以从错误情况逼近。譬如只顾着正着推导，却忘了可以反过来推。只顾着反过来推，居然忘了可以考察简单特例。试了各种手法，却发现忘了考虑题目的某个条件。觉得试遍了所有可能性，已经走不下去了，然后其实在思维的早些时候就已经落入了思维陷阱。等等）事实上，即便是一个熟练的解题者也容易犯顾此失彼的问题，因为我们一旦意识到一个看似能够得到结论的解法，整个注意力就容易被吸引过去，而由于推导的路径是很长的，所以很容易在一条路上走到黑，试图再往下走一步就得出解。却忘了回过头来看看再更高的层面上还有没有其它手法，思路上有没有其他可能性。

  而对于像我这样目前尚不谙熟所有思维方法的人来说，则更容易犯这样的错误。为了避免这样的错误，一个有效的办法就是将自己的思考过程（中的重要环节）清晰的写在纸上（称为“看得见的思考”），这有如下几个好处：

  人在思考一个问题的时候，就像是在黑暗中打着电筒往前走（事实上，我们的工作记忆资源是有限的，有研究证明我们只能在工作记忆里面持有 7 加减 2 个项目；此外认知负荷也是有极限的），每一步推导，每一步逻辑或猜测都将我们往前挪一步，然而电筒的光亮能找到的范围是有限的，我们走了几步发现后面又黑了。有时候，我们是如此努力地试图一下就走出很远，同时又老是怕忘记目前已经取得的进展和重要结论，结果意识的微光就在一个很小的范围内打转，始终无法往前走出很远。而将思维过程记录下来，则给了我们完全的回顾机会。如果你是经常做笔记的人，你肯定会发现，有时候一个在脑子里觉得两句话就能说完不需要记下来的东西，一旦开始往纸上写下来，你就自然而然能得出更多的结论和东西，越写越多，最终关于你的问题的所有方面都被推导出来展现在你面前。

  思考问题时的注意力是自上而下控制大脑的神经处理过程的，当我们集中注意力在某一个过程上时，其它的过程就会受到抑制。我们平常都遇到过一些时候，由于集中注意力从而忽略了周围发生的事情的时候（处理环境输入的神经回路受到抑制）。所以，当我们竭尽全力将

一些非常重要的因素控制在工作记忆里面的时候，实际上很大程度上抑制了其余的思考——可以想见如果科比在跳投的瞬间集中注意力思考跳投的各种技术要素的话会发生怎样的灾难。此外，这么做还占用了宝贵的工作记忆空间，从这个意义上，借助于纸笔，将思考的东西写下来实际上就是扩充了我们的工作记忆，增大了思维的缓存。注意，这倒不是说思考问题不需要集中注意力，而是说由于将项目维持在工作记忆中需要很大的认知精力，使得我们的注意力无法暂时移开去思考其它相关的子问题，而写到纸上的话我们就减轻了工作记忆的负担，可以转移注意力去集中思考某个子问题；同时我们又可以随时回过头来，重新将以前想过的结论装入记忆（内存），完全不用费心去阻止它们被我们的工作记忆遗忘。

一句话从嘴里说出来，或者写到纸上，被视觉或听觉模块接收，再认知；跟在心里默念所产生的神经兴奋程度是不一样的。我们都有过这样的经历：一句令人不愉快的话，我们心里清楚，但就是不愿意自己也不愿意别人从嘴里说出来。同样，将思考的过程写到纸上，能够激起潜在的更多的联想。为什么会这样的另一个可能的原因是我们大脑中思维过程的呈现形式和纸上的表现形式是不一样的，既没有那么严格、详细也没有那么多的符号（如数学符号）——再一次，工作记忆资源是很有限的——而后者，作为视觉线索，可能激起更多对既有知识的回忆。

我们在思考问题的过程中容易落入思维定势，不知不觉就走上来某条“绝大部分时候是如此”的思维捷径，对于一些问题而言这固然能够让我们快速得到解，但对于另一些问题而言却是致命的。我们容易在逻辑的路径上引入想当然的假设，从而排除某种不该排除的可能性或做法。通过将思路过程写到纸上，我们便能够回头细细考察自己的思考过程，觉察到什么地方犯了想当然的毛病。

我们在思维过程中的每一个关键的一步也许都有另一种可能性，一个问题越复杂，需要推导的步骤就越多，我们就越容易忽视过程中的其它可能性，容易一条路走到黑。而将思维过程写下来，在走不下去的时候可以回过头看看，也许会发现另一种可能性，另一条“少有人走的路”。

最后，通过将思维过程写下来，我们就能够在解题完毕之后完整的回顾自己的整个思维过程，并从中再次体悟那些关键的想法背后所发生的心理活动过程，总结思考中的重要的一般原则，分析思维薄弱的环节，等等。就算是最终发现并没有到达结果的无效思路，也未必就没有意义，因为不是因为错误的思路，也不会知道正确的思路，况且对一道题目用不上的思路，对其它题目未必用不上。通过对自己思维过程的彻底反思，就能从每次解题中获得最多的收获。

- 练习，练习 本质上，练习并不产生新能力。然而练习最重要的一个作用就是将外显记忆转化为内隐记忆。用大白话来说就是将平时需要用脑子去想（参与）的东西转化为内在的习惯。譬如我们一开始学骑自行车的时候需要不断提醒自己注意平衡，但随着不断的联系，这种技能就内化成了所谓的程序式记忆（内隐记忆的一种），从而就算你一边骑车一边进行解题这样需要消耗大量脑力的活动，也无需担心失去平衡（不过撞树是完全可能的，但那是另一回事）。

  同样，对于解题中的思维方法来说，不断练习这些思维方法就能做到无意识间就能运用自如，大大降低了意识的负担和加快了解题速度。

  不过，并非所有的练习方法都是等效的，有些练习方法肯定要比另一些更有效率。譬如就解题来说，解题是一项涉及到人类最高级思维机制的活动，其中尤其是推理（归纳和演绎）和联想。而后者中又尤数联想是最麻烦的，前面提到，绝大多数时候启发式方法实质上都是在为联想服务——为了能像晃筛子那样把你脑袋里那个关键的相关知识抖落出来。并且，为了方便以后能够联想，在当初吸收知识的时候就需要做最恰当的加工才行，譬如前面提到的“抽象”加工，除此之外还有将知识与既有的知识框架整合，建立最多的思维连接点（或者说“钩子”）。对于知识的深浅加工所带来的影响，《找寻逝去的自我》里面有精彩的介绍（里面也提到了提取线索对回忆的影响——从该意义上来说运用启发式思维方法来辅助联想，其实就是进行策略性记忆提取的过程）。最后，人类的无意识思维天生有着各种各样的坏习惯，譬如前面提到的范畴陷阱就是创新思维的杀手，譬如根据表面相似性进行类比也是知识转移的一大障碍。更遑论各种各样的思维捷径了（我们平常进行的绝大多数思考和决策，都是通过认知捷径来进行的）。所以说，如果任由我们天生的思维方式发展，也许永远都避不开无意识中的那些陷阱，好在我们除了无意识之外还多出了一层监督机制——意识。通过不断反省思维本身，时时纠正不正确的思考方式，我们就能够对其进行淬炼，最终养成良好的思维习惯。反之被动的练习虽然也能熟能生巧，但势必花的时间更多，而且对于涉及复杂的思维机制的解题活动来说，远远不是通过钱眼往油壶里面倒油这样简单的活动所能类比的，倒油不像思维活动那样有形形色色的陷阱，倒油不需要联想和推理，倒油甚至几乎完全不需要意识的辅助性参与，除了集中注意力（而解题活动就算对于极其熟练的人来说也不断需要大量的意识参与）。所以对于前者，良好的思维习惯至关重要，而反省加上运用正确的思维方法则是最终养成良好思维习惯的途径。

  练习还有另外一个很重要的作用，就是增加领域知识（关于知识在问题解决中的作用，前面已经提到过）。我们看到很多人，拿到一道题目立即脑子里就反应出解法，这个反应快到他自己都不能意识到背后有什么逻辑。这是因为既有的知识（我们常说的“无他，实在是题做得

太多了”）起到了极大的作用，通过对题目中几个关键元素或结构的感知，大脑中的相关知识迅速被自动提取出来。而对于知道但不熟悉相应知识（譬如很早我们就知道归纳法，但是很久以后我们才真正能够做到面对任何一道可能用归纳法的题目就立即能够想到运用归纳法），或者干脆就不知道该知识的人来说，就需要通过启发法来辅助联想或探索了。后者可以一定程度上代偿对知识的不够熟悉，但在一些时候知识的缺失则是致命的（参见上面第 2 点）。不过要注意的是，那种看到题目直接反应出答案的或许也不是纯粹的好事，因为这样的解题过程严重依赖于既有知识，尤其是做过的类似的题目，其思维过程绝大部分运用的是联想或类比，而非演绎或归纳。更重要的是，联想也分两种，被动联想和策略性联想（参考《找寻逝去的自我》），这里用的却是被动联想。所以，能直接反应出答案并不代表遇到真正新颖的题目的时候的解决能力，后者由于不依赖于既有领域知识，就真正需要看一个人的思维能力和习惯究竟如何了。

- 启发法的局限性 首先肯定的是，启发法一定（也许很大）程度上是可以代偿知识的不足的（这里的知识主要是指大脑中的“联系”，下面还会提到另一种知识，即 hard knowledge）。譬如，一道题目，别人直接就能通过类比联想到某道解过的题目，并直接使用了其中的一个关键的性质把题目给解出来了。你并没有做过那道题目，这导致两种可能的结果：一，你就是不知道那个性质。二，你虽然“知道”那个性质，但并没有在以前的解题经历中将那个性质跟你手头的这个问题中的“线索”联系起来，所以你还是“想不到”。后一种可以称为 soft knowledge，即你“知道”，但就是联想（联系）不起来。所谓不能活学活用，某些时候就是这种情况，即书本上提供什么样的知识联系，脑子里也记住什么，而没有事后更广泛地去探索知识之间的本质联系（总结的作用）。前一种则可以称为 hard knowledge，即你就是不知道，它不在你的脑子里。

  而启发式方法在两个层面上起作用：

  辅助联想起 soft knowledge：譬如，特例法是一种启发式思考方法，它通过引入一个简单的特例，特例中往往蕴含有更多的“线索”，通过这些线索，有可能就会激发起对既有的知识的联想。另外一种强大的辅助联想办法就是对题目进行变形，变形之后就产生了新的视觉和语意线索，比如式子的对称性、从直角坐标到极坐标从而引发对后者的知识的联想等等。大量的启发式方法实际上的作用就是辅助联想，通过对题目中的线索的发掘，激起大脑中已知相关知识的浮现。在这个意义上，相对于那些能够直接联想到某个性质的人，那些不知道但可以通过启发式思维联想到的，启发式思维就提供了一种“曲径通幽”的策略性联想。还是以经典的例子来说：砖头的用途。有人立即能够直

接联想到“敲人”。有人也许不能。然而启发式联想策略“抽象”就能够帮助后者也能够联想到“敲人”，因为“抽象”策略启发人去考虑砖头的各个性质维度，如“质地”,“形状”，当你考察到“质地坚硬”,“棱角”，离“敲人”的功能还会远么？本质上，能够直接联想到“敲人”功能的人是因为大脑中从砖头到敲人这两个概念之间的神经通路被走过了很多遍（譬如由于经常拿砖头敲人），神经元之间的联系相当“粗”（形象的说法，严格的事实请参考《追寻记忆的痕迹》），而不经常拿砖头敲人的人呢，这个联系就非常的弱，乃至于根本激不起一次神经冲动。那么为什么通过启发式方法又能联想到呢？因为启发式方法相当于带入了一种新的神经调控回路，首先它增加你联系到砖头的属性维度上的可能性，使得“质地坚硬”、“棱角”这两个语意概念被激活起来（注意，如果没有启发式方法的参与，这是不会发生的），一旦后者被激活起来，从后者到“敲人”的联系就被激活起来了。从本质上，解题中的启发联想方法做的也就是这个工作。而越是一般性的启发式方法就越是能对广泛的问题有帮助（譬如《How to Solve It》中介绍的那些，譬如分类讨论、分治、乃至我认为很重要的一个——写下自己的思维过程，详细分解各个环节，考察思维路径中有无其它可能性（我们很容易拿到一道题目便被一种冲动带入到某一条特定的思路当中，并且遵循着“最可能的”推导路径往下走，往往不自觉的忽略其它可能性，于是那些可能性上的联想就被我们的注意力“抑制”了。））。

辅助探索出 hard knowledge：倒推法是一种启发式思考方法，它将你的注意力集中到问题的结论中蕴含的知识上，一旦你开始关注可能从结论中演绎出来的知识，你就可能得到 hard knowledge，即并不是早先就存在你脑子里，但是可以通过演绎获得的。上文中的最小和子序列中的倒推方法就是一个例子。

而启发式方法的局限性也存在于这两个方面：

有些联系是不管怎样“启发”也想不起来的。譬如“当布被刺破了，干草堆就重要了”，你怎么解释这句话？如果有人提示一下“降落伞”，每个人都会恍然大悟。这是因为从“布”到“降落伞”之间的单向联系是近乎不存在的。而且就算运用启发法，譬如，考虑所有布做的东西，也基本绝无可能想到降落伞，因为同样，从“布做的东西”到“降落伞”之间的关联也是极其微弱的。我们脑子里只能保留那些最最重要的联系。（如果一提到布，“降落伞”和“衣服”、“被单”、“窗帘”等日常物品以同等重要级别闪现，就乱套了。）那为什么从降落伞我们能想到布呢？我们实际上不能，我们为什么有些时候能，是因为譬如有人叫你“考虑降落伞的材料”，后者就激发了“降落伞之材料”这个语意，后者又指导了我们去考察降落伞的材料构成，于是我们想到是布。否则“布”是不会直接被激发起来的。那为什么在我们的这个问题

中，一旦有人提到降落伞，我们就能建立从布到降落伞的关联呢？这是因为“降落伞”和“布”这两个语意单元的同时兴奋增大了它们之间关联的可能性，就好比是加大另一端的电压从而发生了“击穿”一样。从本质上，解数学题也是如此，费马大定理的求解过程是一个很好的例子，谷山志村猜想，就相当于那个“降落伞”的提示。我们还听到很多这样的故事（或者自己经历）：苦思冥想一个问题不得要领，某一天在路上走，看到某个东西或听到某句话，然后忽然，一道闪电划破长空，那个问题解开了（阿基米德是因为躺在浴缸里从而想到浮力原理的吗?）。我敢保证，如果一个人早就把那个问题从脑海里扔到九霄云外去了（不再处于兴奋状态了），那么就算线索出现，也是不可能发生顿悟的。我们都知道，带着一个问题（使其在大脑中处于兴奋状态）去寻找答案更可能找到，即便不是有意去寻找，只要问题还在脑子里，任何周围的有可能与它相关的线索都不会被大脑漏掉，因为“问题”和“周围的其他线索”同时的兴奋增大了关联的可能性。如果问题早就被从大脑（意识或者潜意识）中撤下了，即便周围出现提示也不会被捕捉到。

许多 hard knowledge 是不能被启发探索出来的。至少是不能被“直接命中目标”地探索出来的。一个问题有可能跟三角函数有关，也许你只能带着问题去探索三角函数的所有性质，从而最终发现那个关键的性质。费马大定理与椭圆方程有关，也许只能去探索椭圆方程的所有性质，这个过程一定程度上是盲目的，试错的，遍历的。而不是直接面向目标的。再聪明的人也无法从费马大定理直接反推到谷山志村猜想。在这些时候，启发式方法最多只能提供一个探索的大致方向：譬如，探索三角函数的性质，并随时注意其中哪个可能对我这个问题有帮助。譬如，探索模运算的性质，看看哪些性质可能会有用。譬如，探索椭圆曲线的性质…等等。启发式方法并不能使我们的探索精准地命中目标。而只能划定一个大致的范围。也难怪有人说数学是盲目的。

但话说回来，启发式方法的局限性并不能否认在大量场合启发式方法的巨大帮助，许多时候，单靠启发式方法就能带来突破。而且，一旦知识性的东西掌握的是一样多的，能否运用更优秀的思维方法就决定了能力的高下。有很多介绍思维方法的书。

- 总结的意义 解题练习的最重要目的不是将特定的题目解出来，而是在于反思解题过程中的一般性的，跨问题的思维法则。简单的将题目解出来（或者解不出来看答案，然后“恍然大悟”），只能得到最少的东西，解出来固然能够强化导致解出来的那个思维过程和方法，但缺少反思的话便不能抽取出一般性的东西供更多的题目所用。而解不出来，看答案然后“哦”的一声更是等同于没有收获，因为“理解”和“运用”相差何止十万八千里。每个人都有过这样的经历：一道题目苦思

冥想不得要领，经某个人一指点其中的关键一步，顿时恍然大悟——这是理解。但这个理解是因为别人已经将新的知识（那个关键的一步）放到你脑子里了，故而你才能理解。而要运用的话，则需要自己去想出那关键的一步。因此，去揣测和总结别人的思维是如何触及那关键的一步，而你自己的思维又为什么触及不到它，有一些一般性的原则可以指导你下次也能想到那个“关键的一步”吗，是很有意义的。我们很多时候会发现，一道题目，解不出来，最终在提示下面解出来之后，发现其中并没有用到任何自己不知道的知识，那么不仅就要问，既然那个知识是在脑子里的，为什么我们当时愣是提取不出来呢？而为什么别人又能够提取出来呢？我怎么才能像别人那样也提取出相应的知识呢？实际上这涉及到关于记忆的最深刻的原理，实际上文中已经提到了一些。（有兴趣的建议参考以下几本书：《追寻记忆的痕迹》，《找寻逝去的自我》，《Synaptic Self》，《Psychology of Problem Solving》）一般性的思维法则除了对于辅助联想（起关键的知识）之外，另一个作用就是辅助演绎/归纳（助探），一开始学解题的时候，我们基本上是先读懂题目条件，做可能的一些显然的演绎。如果还没推到答案的话，基本就只能愣在那里等着那个关键的步骤从脑子里冒出来了。而所谓的启发式思维方法，就是在这个时候可以运用一些一般性的，所有题目都适用的探索手法，进一步去探索问题中蕴含的知识，从而增大成功解题的可能性。启发式的思维方法有很多，从一般到特殊，最具一般性的，在波利亚的《How to Solve It》中已经基本全部都介绍了。一些更为特殊性的（譬如“如果全局搜索空间没有递归结构，那么考虑分割搜索空间”，譬如那些“看到 XX，要想到 YY”的联系），则需要自己在练习中不断抽象总结。

《算法》书评

这是本很新的书，06 年末发行，07 年才慢慢出现于人们的视野。我在 08 年初得知这本书，那会我还很奇怪：都什么年月了，怎么还有人写算法教材——这么“经典”的工作，不是上个世纪就被人做完了吗。

读了这本 Algorithms，我才知道：这才是我心中的算法书，我等待这样一本书已经很多年了。它的确当得起这个名字。

书的三位作者：Sanjoy Dasgupta, Papadimitriou, Umesh Vazirani。

其中，Umesh 堪称计算机理论界的第二名师（第一名师是他自己的导师 Manuel Blum），他带过的学生们犹如一个理论计算机科学新生代的全明星队。另一个作者 Papadimitriou 可算是理论界的第二名笔（第一非 Knuth 莫属），他的书 Computational Complexity 和 Combinatorial optimization 堪称理论计算机科学最好读的专业书，他业余还写了本小说”Turing”。第三个作者 Dasgupta 是个算法方向的研究者，他最年轻，本身就是 Umesh 的学生，相比前面二位也没什么噱头——可他注定要因这本 Algorithms 而被载

入计算机科学的史册。

在这本书之前，算法的经典教材首推 CLRS 的算法导论。算法导论让人印象深刻的，是它内容的全面翔实，还有它一千两百页的厚度。

而见到这本 Algorithms 时，你会震惊于它的薄。我从亚马逊收到这本书时，还以为拿错了包裹。

可读过之后，你就会折服于它的美。

这是一本可以给人带来巨大阅读乐趣的专业书籍。作者娓娓道来，又惜墨如金。用极精炼的语言，为我们指明了一条通向那些美丽算法的线索。我要由衷地说：这本书不仅仅是一些结果的集合，更是一段美好的旅程。我对书中涉及的内容已然熟悉，但读过之后仍感收获良多，对算法这门学问又多了些认识。真的是，写书当如是。

对我来说，算法的教与学有两个困难的地方：

其一，我们学习了那些经典的算法，除了赞叹一下设计的巧思，但总难免问上一句：怎么想到的？对学生来说，这可能是最费解、也最让人窝火的地方。我们下再多的功夫去记忆书上的算法、去分析这些算法的效率，却终究不能理喻得到这些算法的过程。心理盘算着：给我一个新问题，让我设计个算法出来，我能行吗？答案是：不知道。

可这偏偏又是极重要的，无论作研究还是实际工作，一个计算机专业人士最重要的能力，就是解决问题——解决那些不断从理论模型、或者从实际应用中冒出来的新问题。

其二，算法作为一门学问，有两条正交的线索。一个是算法处理的对象：数、矩阵、集合、串 (strings)、排列 (permutations)、图 (graphs)、表达式 (formula)、分布 (distributions)，等等。另一个是算法的设计思想：贪婪、分治、动态规划、线性规划、局部搜索 (local search)，等等。这两条线索几乎是相互独立的：同一个离散对象，例如图，稍有不同的问题，例如 single-source shortest path 和 all-pair shortest path，就可以用到不同的设计思想，如贪婪和动态规划；而完全不同的离散对象上的问题，例如排序和整数乘法，也许就会用到相同的思想，例如分治。

两条线索交织在一起，该如何表述。对学生而言，不同离散对象的差别是直观的——我们已经惯于在一章里面完全讲排序、而在另一章里面完全讲图论算法；可是对算法而言，设计思想的差别是根本的，因为这是从问题的结构来的：不同离散对象上面定义的问题，可以展现出类似的结构，而这结构特征，就是支持一类算法的根本，也是我们设计算法的依据。

坦率的说，之前还没有哪一本算法书很好的解决这两个困难，就连算法导论在这两个问题上也都做得不好。传统的算法书，大多注重内容 (content) 的收录，但却忽视思维过程的展示 (exposition)，因此我们学习了经典的算法，却费解于得到算法的过程；而且算法教材对于内容的编排多是枚举式的 (enumerative)，这多少是受了 the art of computer programming 的影响——可那是本工具书而不是教材，因此我们一提到算法课，就想起了排序、哈希、最短路径……这些题目 (topics)，却没有一个统一的线索在心中。

这本 Algorithms，在短短的篇幅内，做到了。

三位作者可谓野心勃勃，几乎是胆大妄为。他们对传统算法教学思路的颠覆和背叛可谓前所未有。刚拿到目录的时候，我就替他们捏了一把汗，觉得这哪里像一本“正经”的算法书。可读下来，却不由得佩服——算法书早该这么写了。

他们并没有要全面的收录各种各样的算法，他们做的事情是理清了一条算法这门学问的线索。因此填鸭式的内容灌输不是这本书的目的；对结构的精心安排，对问题的数学结构的剖析、从而推出一个算法的过程的讲解，都体现除了这本书真正的用心：它要让学生获得最大程度的启发，要训练学生独立解决问题的能力。

我觉得这才是教育的真正目的，也是算法课应该追求的目标。

说完了种种溢美之词，也来补充一下这本书的不足。这样一本精炼的算法书，为了它道理的清晰、为了它的美，必然会放弃一点对面面俱到的追求。如果我用这本书来教一门算法课的话，我会增加一点以下的内容：

1. 数据结构。

这本书对数据结构没有单独的章节，都是在某个数据结构被一个算法用到的时候讲一下，例如 priority queue 之于 Dijkstra's algorithm。这种做法体现了作者的观点：这门课完全就是关于 algorithms，数据结构对于算法而言就只是个工具。如果同一个系还能开出一门很强的 data structures 课，这么做当然很好，各有侧重。但若是我来上课，肯定会提一下数据结构的一些重要思想，例如 hashing，和他们的数学背景。因为对于一些实际问题，数据结构已不再是个工具，可能就是问题本身。

1. 几个没有被此书涉及到的算法设计和分析的工具：对手论证 (adversarial argument)，matroid，平摊分析 (amortized analysis)。

2. 书中每个算法问题目前最好的上下界 (upper bounds, lower bounds)。

对于一本书而言，让它记录这些不太现实，因为除非上下界已经紧了，也许出版的第二天就会有更好的上界或下界（其实这事已经发生了，书最结尾 historical notes 提了一句整数乘法的 fastest known algorithm，结果现在这个结果已经被刷新了）。但老师上课的时候，应该跟学生们讲一下这个内容，让学生心里有这个“上下界”和”open problem” 的概念，也晓得算法不是死知识，而是正在发生中的事。

1. 讲一点比贪婪、动态规划等等更加“现代”的算法设计的思想，例如 spectral analysis, metric embedding, rapid mixing of markov chain 等等。

也提一点当下的实际问题（例如 google 或豆瓣想解决的问题）面临的一些新的考虑，例如非经典的现实的计算模型：考虑内外存的 I/O 模型，面对海量数据输入的 streaming model，海量数据的 sub-linear algorithms 等等。这些现实模型上的算法需要重新设计去面对新的考量。我理解 Algorithms 这本书没有收录这些内容的原因。越是新的东西老的越快，没有人希望自己的书很快过时。但上课不妨出一些这样的 case studies，时髦的东西学生肯定会很感兴趣，这些新东西的粗糙也可以锻炼学生实际解决问题的能力。

1. 除了这本 Algorithms 作为教材，补充读物可以在 CLRS 算法导论和 Kleinberg 和 Tardos 的算法设计（这也是本顶新的书）之间选择一本。我个人推荐后者。

三国志卷三十六五虎将列传

- 關羽 關羽字雲長，本字長生，河東解人也。亡命奔涿郡。先主於鄉里合徒眾，而羽與張飛為之禦侮。先主為平原相，以羽、飛為別部司馬，分統部曲。先主與二人寢則同床，恩若兄弟。而稠人廣坐，侍立終日，隨先主周旋，不避艱險。〈蜀記曰：曹公與劉備圍呂布於下邳，關羽啟公，布使秦宜祿行求救，乞娶其妻，公許之。臨破，又屢啟於公。公疑其有異色，先遣迎看，因自留之，羽心不自安。此與魏氏春秋所說無異也。〉先主之襲殺徐州刺史車冑，使羽守下邳城，行太守事，〈魏書云：以羽領徐州。〉而身還小沛。

  建安五年，曹公東征，先主奔袁紹。曹公禽羽以歸，拜為偏將軍，禮之甚厚。紹遣大將軍顏良攻東郡太守劉延於白馬，曹公使張遼及羽為先鋒擊之。羽望見良麾蓋，策馬刺良於萬眾之中，斬其首還，紹諸將莫能當者，遂解白馬圍。曹公即表封羽為漢壽亭侯。初，曹公壯羽為人，而察其心神無久留之意，謂張遼曰：「卿試以情問之。」既而遼以問羽，羽歎曰：「吾極知曹公待我厚，然吾受劉將軍厚恩，誓以共死，不可背之。吾終不留，吾要當立效以報曹公乃去。」遼以羽言報曹公，曹公義之。〈傅子曰：遼欲白太祖，恐太祖殺羽，不白，非事君之道，乃歎曰：「公，君父也；羽，兄弟耳。」遂白之。太祖曰：「事君不忘其本，天下義士也。度何時能去?」遼曰：「羽受公恩，必立效報公而後去也。」〉及羽殺顏良，曹公知其必去，重加賞賜。羽盡封其所賜，拜書告辭，而奔先主於袁軍。左右欲追之，曹公曰：「彼各為其主，勿追也。」〈臣松之以為曹公知羽不留而心嘉其志，去不遣追以成其義，自非有王霸之度，孰能至於此乎？斯實曹公之休美。〉

  從先主就劉表。表卒，曹公定荊州，先主自樊將南渡江，別遣羽乘船數百艘會江陵。曹公追至當陽長阪，先主斜趣漢津，適與羽船相值，共至夏口。〈蜀記曰：初，劉備在許，與曹公共獵。獵中，眾散，羽勸備殺公，備不從。及在夏口，飄颻江渚，羽怒曰：「往日獵中，若從羽

言，可無今日之困。」備曰：「是時亦為國家惜之耳；若天道輔正，安知此不為福邪！」臣松之以為備後與董承等結謀，但事泄不克諧耳，若為國家惜曹公，其如此言何！羽若果有此勸而備不肯從者，將以曹公腹心親戚，實繁有徒，事不宿構，非造次所行；曹雖可殺，身必不免，故以計而止，何惜之有乎！既往之事，故讬為雅言耳。〉孫權遣兵佐先主拒曹公，曹公引軍退歸。先主收江南諸郡，乃封拜元勳，以羽為襄陽太守、蕩寇將軍，駐江北。先主西定益州，拜羽董督荊州事。羽聞馬超來降，舊非故人，羽書與諸葛亮，問超人才可誰比類。亮知羽護前，乃答之曰：「孟起兼資文武，雄烈過人，一世之傑，黥、彭之徒，當與益德並驅爭先，猶未及髯之絕倫逸群也。」羽美鬚髯，故亮謂之髯。羽省書大悅，以示賓客。

羽嘗為流矢所中，貫其左臂，後創雖愈，每至陰雨，骨常疼痛，醫曰：「矢鏃有毒，毒入於骨，當破臂作創，刮骨去毒，然後此患乃除耳。」羽便伸臂令醫劈之。時羽適請諸將飲食相對，臂血流離，盈於盤器，而羽割炙引酒，言笑自若。

二十四年，先主為漢中王，拜羽為前將軍，假節鉞。是歲，羽率眾攻曹仁於樊。曹公遣于禁助仁。秋，大霖雨，漢水汎溢，禁所督七軍皆沒。禁降羽，羽又斬將軍龐德。梁、郟、陸渾群盜或遙受羽印號，為之支黨，羽威震華夏。曹公議徙許都以避其銳，司馬宣王、蔣濟以為關羽得志，孫權必不原也。可遣人勸權躡其後，許割江南以封權，則樊圍自解。曹公從之。先是，權遣使為子索羽女，羽罵辱其使，不許婚，權大怒。〈典略曰：羽圍樊，權遣使求助之，敕使莫速進，又遣主簿先致命於羽。羽忿其淹遲，又自已得於禁等，乃罵曰：「鯫子敢爾，如使樊城拔，吾不能滅汝邪！」權聞之，知其輕己，偽手書以謝羽，許以自往。臣松之以為荊、吳雖外睦，而內相猜防，故權之襲羽，潛師密發。按呂蒙傳雲：「伏精兵於　蒋　鹿之中，使白衣搖櫓，作商賈服。」以此言之，羽不求助於權，權必不語羽當往也。若許相援助，何故匿其形跡乎?〉又南郡太守麋芳在江陵，將軍士仁屯公安，素皆嫌羽輕己。羽之出軍，芳、仁供給軍資，不悉相救。羽言「還當治之」，芳、仁鹹懷懼不安。於是權陰誘芳、仁，芳、仁使人迎權。而曹公遣徐晃救曹仁，〈蜀記曰：羽與晃宿相愛，遙共語，但說平生，不及軍事。須臾，晃下馬宣令：「得關雲長頭，賞金千斤。」羽驚怖，謂晃曰：「大兄，是何言邪！」晃曰：「此國之事耳。」〉羽不能克，引軍退還。權已據江陵，盡虜羽士眾妻子，羽軍遂散。權遣將逆擊羽，斬羽及子平於臨沮。〈蜀記曰：權遣將軍擊羽，獲羽及子平。權欲活羽以敵劉、曹，左右曰：「狼子不可養，後必為害。曹公不即除之，自取大患，乃議徙都。今豈可生！」乃斬之。臣松之按吳書：孫權遣將潘璋逆斷羽走路，羽至即斬，且臨沮去江陵二三百里，豈容不時殺羽，方議其生死乎？又云「權欲活羽以敵劉、曹」，此之不然，可以絕智者之口。吳曆曰：權送羽首於曹公，以

諸侯禮葬其屍骸。〉

追諡羽曰壯繆侯。〈蜀記曰：羽初出軍圍樊，夢豬齧其足，語子平曰：「吾今年衰矣，然不得還！」江表傳曰：羽好左氏傳，諷誦略皆上口。〉子興嗣。興字安國，少有令問，丞相諸葛亮深器異之。弱冠為侍中、中監軍，數歲卒。子統嗣，尚公主，官至虎賁中郎將。卒，無子，以興庶子彝續封。〈蜀記曰：龐德子會，隨鍾、鄧伐蜀，蜀破，盡滅關氏家。〉

- 張飛　　張飛字益德，涿郡人也，少與關羽俱事先主。羽年長數歲，飛兄事之。先主從曹公破呂布，隨還計，曹公拜飛為中郎將。先主背曹公依袁紹、劉表。表卒，曹公入荊州，先主奔江南。曹公追之，一日一夜，及於當陽之長阪。先主聞曹公卒至，棄妻子走，使飛將二十騎拒後。飛據水斷橋，瞋目橫矛曰：「身是張益德也，可來共決死！」敵皆無敢近者，故遂得免。先主既定江南，以飛為宜都太守、徵虜將軍，封新亭侯，後轉在南郡。先主入益州，還攻劉璋，飛與諸葛亮等溯流而上，分定郡縣。至江州，破璋將巴郡太守嚴顏，生獲顏。飛呵顏曰：「大軍至，何以不降而敢拒戰？」顏答曰：「卿等無狀，侵奪我州，我州但有斷頭將軍，無有降將軍也。」飛怒，令左右牽去斫頭，顏色不變，曰：「斫頭便斫頭，何為怒邪！」飛壯而釋之，引為賓客。〈華陽國志曰：初，先主入蜀，至巴郡，顏拊心歎曰：「此所謂獨坐窮山，放虎自衛也！」〉飛所過戰克，與先主會於成都。益州既平，賜諸葛亮、法正、飛及關羽金各五百斤，銀千斤，錢五千萬，錦千匹，其餘頒賜各有差，以飛領巴西太守。

曹公破張魯，留夏侯淵、張郃守漢川。郃別督諸軍下巴西，欲徙其民於漢中，進軍宕渠、蒙頭、蕩石，與飛相拒五十餘日。飛率精卒萬餘人，從他道邀郃軍交戰，山道迮狹，前後不得相救，飛遂破郃。郃棄馬緣山，獨與麾下十餘人從間道退，引軍還南鄭，巴土獲安。先主為漢中王，拜飛為右將軍、假節。章武元年，遷車騎將軍，領司隸校尉，進封西鄉侯，策曰：「朕承天序，嗣奉洪業，除殘靖亂，未燭厥理。今寇虜作害，民被荼毒，思漢之士，延頸鶴望。朕用怛然，坐不安席，食不甘味，整軍誥誓，將行天罰。以君忠毅，侔蹤召、虎，名宣遐邇，故特顯命，高墉進爵，兼司於京。其誕將天威，柔服以德，伐叛以刑，稱朕意焉。詩不雲乎，’匪疚匪棘，王國來極。肇敏戎功，用錫爾祉’。可不勉歟！」

初，飛雄壯威猛，亞於關羽，魏謀臣程昱等咸稱羽、飛萬人之敵也。羽善待卒伍而驕於士大夫，飛愛敬君子而不恤小人。先主常戒之曰：「卿刑殺既過差，又日鞭撻健兒，而令在左右，此取禍之道也。」飛猶不悛。先主伐吳，飛當率兵萬人，自閬中會江州。臨發，其帳下將張達、范強殺飛，持其首，順流而奔孫權。飛營都督表報先主，先主聞

飛都督之有表也，曰：「噫！飛死矣。」追諡飛曰桓侯。長子苞，早夭。次子紹嗣，官至侍中尚書僕射。苞子遵為尚書，隨諸葛瞻於綿竹，與鄧艾戰，死。

- 馬超　　馬超字孟起，右扶風茂陵人也。父騰，靈帝末與邊章、韓遂等俱起事於西州。初平三年，遂、騰率眾詣長安。漢朝以遂為鎮西將軍，遣還金城，騰為征西將軍，遣屯郿。後騰襲長安，敗走，退還涼州。司隸校尉鍾繇鎮關中，移書遂、騰，為陳禍福。騰遣超隨繇討郭援、高幹於平陽，超將龐德親斬援首。後騰與韓遂不和，求還京畿。於是徵為衛尉，以超為偏將軍，封都亭侯，領騰部曲。〈典略曰：騰字壽成，馬援後也。桓帝時，其父字子碩，嘗為天水蘭幹尉。後失官，因留隴西，與羌錯居。家貧無妻，遂娶羌女，生騰。騰少貧無產業，常從彰山中斫材木，負販詣城市，以自供給。騰為人長八尺餘，身體洪大，面鼻雄異，而性賢厚，人多敬之。靈帝末，涼州刺史耿鄙任信奸吏，民王國等及氐、羌反叛。州郡募發民中有勇力者，欲討之，騰在募中。州郡異之，署為軍從事，典領部眾。討賊有功，拜軍司馬，後以功遷偏將軍，又遷徵西將軍，常屯汧、隴之間。初平中，拜徵東將軍。是時，西州少穀，騰自表軍人多乏，求就谷於池陽，遂移屯長平岸頭。而將王承等恐騰為己害，乃攻騰營。時騰近出無備，遂破走，西上。會三輔亂，不復來東，而與鎮西將軍韓遂結為異姓兄弟，始甚相親，後轉以部曲相侵入，更為讎敵。騰攻遂，遂走，合眾還攻騰，殺騰妻子，連兵不解。建安之初，國家綱紀殆弛，乃使司隸校尉鍾繇、涼州牧韋端和解之。徵騰還屯槐裏，轉拜為前將軍，假節，封槐裏侯。北備胡寇，東備白騎，待士進賢，矜救民命，三輔甚安愛之。十（五）年，徵為衛尉，騰自見年老，遂入宿衛。初，曹公為丞相，辟騰長子超，不就。超後為司隸校尉督軍從事，討郭援，為飛矢所中，乃以囊囊其足而戰，破斬援首。詔拜徐州刺史，後拜諫議大夫。及騰之入，因詔拜為偏將軍，使領騰營。又拜超弟休奉車都尉，休弟鐵騎都尉，徙其家屬皆詣鄴，惟超獨留。〉

超既統眾，遂與韓遂合從，及楊秋、李堪、成宜等相結，進軍至潼關。曹公與遂、超單馬會語，超負其多力，陰欲突前捉曹公，曹公左右將許褚瞋目盼之，超乃不敢動。曹公用賈詡謀，離間超、遂，更相猜疑，軍以大敗。〈山陽公載記曰：初，曹公軍在蒲阪，欲西渡，超謂韓遂曰：「宜於渭北拒之，不過二十日，河東穀盡，彼必走矣。」遂曰：「可聽令渡，蹙於河中，顧不快耶！」超計不得施。曹公聞之曰：「馬兒不死，吾無葬地也。」〉超走保諸戎，曹公追至安定，會北方有事，引軍東還。楊阜說曹公曰：「超有信、布之勇，甚得羌、胡心。若大軍還，不嚴為其備，隴上諸郡非國家之有也。」超果率諸戎以擊隴上郡縣，隴上郡縣皆應之，殺涼州刺史韋康，據冀城，有其眾。超自稱徵西將軍，領並州牧，督涼州軍事。康故吏民楊阜、姜敘、梁寬、趙衢等，合謀擊

超。阜、敘起於鹵城，超出攻之，不能下；寬、衢閉冀城門，超不得入。進退狼狽，乃奔漢中依張魯。魯不足與計事，內懷於邑，聞先主圍劉璋於成都，密書請降。〈典略曰：建安十六年，超與關中諸將侯選、程銀、李堪、張橫、梁興、成宜、馬玩、楊秋、韓遂等，凡十部，俱反，其眾十萬，同據河、潼，建列營陳。是歲，曹公西徵，與超等戰於河、渭之交，超等敗走。超至安定，遂奔涼州。詔收滅超家屬。超複敗於隴上。後奔漢中，張魯以為都講祭酒，欲妻之以女，或諫魯曰：「有人若此不愛其親，焉能愛人?」魯乃止。初，超未反時，其小婦弟種留三輔，及超敗，種先入漢中。正旦，種上壽於超，超搥胸吐血曰：「闔門百口，一旦同命，今二人相賀邪?」後數從魯求兵，欲北取涼州，魯遣往，無利。又魯將楊白等欲害其能，超遂從武都逃入氐中，轉奔往蜀。是歲建安十九年也。〉

先主遣人迎超，超將兵徑到城下。城中震怖，璋即稽首，〈典略曰：備聞超至，喜曰：「我得益州矣。」乃使人止超，而潛以兵資之。超到，令引軍屯城北，超至未一旬而成都潰。〉以超為平西將軍，督臨沮，因為前都亭侯。〈山陽公載記曰：超因見備待之厚，與備言，常呼備字，關羽怒，請殺之。備曰：「人窮來歸我，卿等怒，以呼我字故而殺之，何以示於天下也!」張飛曰：「如是，當示之以禮。」明日大會，請超入，羽、飛並杖刀立直，超顧坐席，不見羽、飛，見其直也，乃大驚，遂一不復呼備字。明日歎曰：「我今乃知其所以敗。為呼人主字，幾為關羽、張飛所殺。」自後乃尊事備。臣松之按以為超以窮歸備，受其爵位，何容傲慢而呼備字？且備之入蜀，留關羽鎮荊州，羽未嘗在益土也。故羽聞馬超歸降，以書問諸葛亮「超人才可誰比類」，不得如書所雲。羽焉得與張飛立直乎？凡人行事，皆謂其可也，知其不可，則不行之矣。超若果呼備字，亦謂於理宜爾也。就令羽請殺超，超不應聞，但見二子立直，何由便知以呼字之故，雲幾為關、張所殺乎？言不經理，深可忿疾也。袁暐、樂資等諸所記載，穢雜虛謬，若此之類，殆不可勝言也。〉先主為漢中王，拜超為左將軍，假節。章武元年，遷驃騎將軍，領涼州牧，進封鄠鄉侯，策曰：「朕以不德，獲繼至尊，奉承宗廟。曹操父子，世載其罪，朕用慘怛，疢如疾首。海內怨憤，歸正反本，暨於氐、羌率服，獯鬻慕義。以君信著北土，威武並昭，是以委任授君，抗颺虓虎，兼董萬里，求民之瘼。其明宣朝化，懷保遠邇，肅慎賞罰，以篤漢祜，以對於天下。」二年卒，時年四十七。臨沒上疏曰：「臣門宗二百餘口，為孟德所誅略盡，惟有從弟岱，當為微宗血食之繼，深讬陛下，餘無複言。」追諡超曰威侯，子承嗣。岱位至平北將軍，進爵陳倉侯。超女配安平王理。〈典略曰：初超之入蜀，其庶妻董及子秋，留依張魯。魯敗，曹公得之，以董賜閻圃，以秋付魯，魯自手殺之。〉

- 黃忠　　黃忠字漢升，南陽人也。荊州牧劉表以為中郎將，與表從子磐共守長沙攸縣。及曹公克荊州，假行裨將軍，仍就故任，統屬長沙

守韓玄。先主南定諸郡，忠遂委質，隨從入蜀。自葭萌受任，還攻劉璋，忠常先登陷陣，勇毅冠三軍。益州既定，拜為討虜將軍。建安二十四年，於漢中定軍山擊夏侯淵。淵眾甚精，忠推鋒必進，勸率士卒，金鼓振天，歡聲動谷，一戰斬淵，淵軍大敗。遷征西將軍。是歲，先主為漢中王，欲用忠為後將軍，諸葛亮說先主曰：「忠之名望，素非關、馬之倫也。而今便令同列。馬、張在近，親見其功，尚可喻指；關遙聞之，恐必不悅，得無不可乎！」先主曰：「吾自當解之。」遂與羽等齊位，賜爵關內侯。明年卒，追諡剛侯。子敘，早沒，無後。

- 趙雲　　趙雲字子龍，常山真定人也。本屬公孫瓚，瓚遣先主為田楷拒袁紹，雲遂隨從，為先主主騎。〈雲別傳曰：雲身長八尺，姿顏雄偉，為本郡所舉，將義從吏兵詣公孫瓚。時袁紹稱冀州牧，瓚深憂州人之從紹也，善雲來附，嘲雲曰：「聞貴州人皆原袁氏，君何獨回心，迷而能反乎？」雲答曰：「天下訩訩，未知孰是，民有倒縣之厄，鄙州論議，從仁政所在，不為忽袁公私明將軍也。」遂與瓚征討。時先主亦依讬瓚，每接納雲，雲得深自結讬。雲以兄喪，辭瓚暫歸，先主知其不反，捉手而別，雲辭曰：「終不背德也。」先主就袁紹，雲見於鄴。先主與雲同床眠臥，密遣雲合募得數百人，皆稱劉左將軍部曲，紹不能知。遂隨先主至荊州。〉及先主為曹公所追於當陽長阪，棄妻子南走，雲身抱弱子，即後主也，保護甘夫人，即後主母也，皆得免難。遷為牙門將軍。先主入蜀，雲留荊州。〈雲別傳曰：初，先主之敗，有人言雲已北去者，先主以手戟擿之曰：「子龍不棄我走也。」頃之，雲至。從平江南，以為偏將軍，領桂陽太守，代趙範。範寡嫂曰樊氏，有國色，範欲以配雲。雲辭曰：「相與同姓，卿兄猶我兄。」固辭不許。時有人勸雲納之，雲曰：「範迫降耳，心未可測；天下女不少。」遂不取。範果逃走，雲無纖介。先是，與夏侯惇戰於博望，生獲夏侯蘭。蘭是雲鄉里人，少小相知，雲白先主活之，薦蘭明於法律，以為軍正。雲不用自近，其慎慮類如此。先主入益州，雲領留營司馬。此時先主孫夫人以權妹驕豪，多將吳吏兵，縱橫不法。先主以雲嚴重，必能整齊，特任掌內事。權聞備西徵，大遣舟船迎妹，而夫人內欲將後主還吳，雲與張飛勒兵截江，乃得後主還。〉

　　先主自葭萌還攻劉璋，召諸葛亮。亮率雲與張飛等俱溯江西上，平定郡縣。至江州，分遣雲從外水上江陽，與亮會於成都。成都既定，以雲為翊軍將軍。〈雲別傳曰：益州既定，時議欲以成都中屋舍及城外園地桑田分賜諸將。雲駁之曰：「霍去病以匈奴未滅，無用家為，令國賊非但匈奴，未可求安也。須天下都定，各反桑梓，歸耕本土，乃其宜耳。益州人民，初罹兵革，田宅皆可歸還，今安居複業，然後可役調，得其歡心。」先主即從之。夏侯淵敗，曹公爭漢中地，運米北山下，數千萬囊。黃忠以為可取，雲兵隨忠取米。忠過期不還，雲將數十騎輕行出圍，迎視忠等。值曹公揚兵大出，雲為公前鋒所擊，方戰，其大眾

至，勢偪，遂前突其陳，且鬥且卻。公軍散，已復合，雲陷敵，還趣圍。將張著被創，雲復馳馬還營迎著。公軍追至圍，此時沔陽長張翼在雲圍內，翼欲閉門拒守，而雲入營，更大開門，偃旗息鼓。公軍疑雲有伏兵，引去。雲雷鼓震天，惟以戎弩於後射公軍，公軍驚駭，自相蹂踐，墮漢水中死者甚多。先主明旦自來至雲營圍視昨戰處，曰：「子龍一身都是膽也。」作樂飲宴至暝，軍中號雲為虎威將軍。孫權襲荊州，先主大怒，欲討權。雲諫曰：「國賊是曹操，非孫權也，且先滅魏，則吳自服。操身雖斃，子丕篡盜，當因眾心，早圖關中，居河、渭上流以討凶逆，關東義士必裹糧策馬以迎王師。不應置魏，先與吳戰；兵勢一交，不得卒解。」先主不聽，遂東征，留雲督江州。先主失利於秭歸，雲進兵至永安，吳軍已退。〉建興元年，為中護軍、征南將軍，封永昌亭侯，遷鎮東將軍。五年，隨諸葛亮駐漢中。明年，亮出軍，揚聲由斜谷道，曹真遣大眾當之。亮令雲與鄧芝往拒，而身攻祁山。雲、芝兵弱敵強，失利於箕谷，然斂眾固守，不至大敗。軍退，貶為鎮軍將軍。〈雲別傳曰：亮曰：「街亭軍退，兵將不復相錄，箕穀軍退，兵將初不相失，何故?」芝答曰：「雲身自斷後，軍資什物，略無所棄，兵將無緣相失。」雲有軍資餘絹，亮使分賜將士，雲曰：「軍事無利，何為有賜？其物請悉入赤岸府庫，須十月為冬賜。」亮大善之。〉

七年卒，追謚順平侯。

初，先主時，惟法正見謚；後主時，諸葛亮功德蓋世，蔣琬、費禕荷國之重，亦見謚；陳祇寵待，特加殊獎，夏侯霸遠來歸國，故複得謚；於是關羽、張飛、馬超、龐統、黃忠及雲乃追謚，時論以為榮。〈雲別傳載後主詔曰：「雲昔從先帝，功積既著。朕以幼沖，涉塗艱難，賴恃忠順，濟於危險。夫謚所以敘元勳也，外議雲宜謚。」大將軍姜維等議，以為雲昔從先帝，勞績既著，經營天下，遵奉法度，功效可書。當陽之役，義貫金石，忠以衛上，君念其賞，禮以厚下，臣忘其死。死者有知，足以不朽；生者感恩，足以殞身。謹按謚法，柔賢慈惠曰順，執事有班曰平，克定禍亂曰平，應謚雲曰順平侯。〉雲子統嗣，官至虎賁中郎，督行領軍。次子廣，牙門將，隨姜維遝中，臨陳戰死。

評曰：關羽、張飛皆稱萬人之敵，為世虎臣。羽報效曹公，飛義釋嚴顏，並有國士之風。然羽剛而自矜，飛暴而無恩，以短取敗，理數之常也。馬超阻戎負勇，以覆其族，惜哉！能因窮致泰，不猶愈乎！黃忠、趙雲強摯壯猛，並作爪牙，其灌、滕之徒歟？

4.20 Final Fantasy Series

comments

作者:苦艾酒链接:https://www.zhihu.com/question/29902814/answer/73130569 来源：知乎著作权归作者所有。商业转载请联系作者获得授权，非

商业转载请注明出处。

作为玩过除了 11 和 14 以外所有最终幻想的玩家，我一般觉得 6,7,9 这三款作品的素质明显超过其他作品。

7 我觉得像是一部高明的侦探小说，明线暗线，盘根错节，导演用剥茧抽丝的手法一点一点解开真相，到谜底摊开的时候让人不由得不拍案叫绝，整体故事荡气回肠。

6 更像是一出出场景剧，每个章节都个性鲜明，表现力不同寻常，让人不管从哪处剧情切进去看都能感受到 6 那种每一个音符都能爆发出最强音的那种旺盛的生命力。

而对我来说，9 就像是一盘大厨做出来的无懈可击的小葱拌豆腐，明明一清二白，却其中味道的层次感，调理的清晰程度，你越咂摸越能感觉到 9 的精髓。9 不需要华丽光鲜的外表，也没有刻意追求每一个场景都让你印象深刻，角色，剧情，音乐浑然一体，让人完全看不出破绽，那种功力真的是只有“见过天地”的人才能理解和追捧的。9 对于生命意义的思考，在我看来已经超越了电子游戏的境界，就算说是心理学教授写出来的我也能相信，就把 Vivi 的独白当做我对 9 代最美好的回忆吧：

I always talked about you, Zidane. How you were a very special person to us, because you taught us all how important life is. You taught me that life doesn't last forever. That's why we have to help each other and live life to the fullest. Even if you say goodbye, you'll always be in our hearts. So, I know we're not alone anymore. Why I was born… How I wanted to live… Thanks for giving me time to think. To keep doing what you set your heart on… It's a very hard thing to do. We were all so courageous… What to do when I felt lonely… That was the only thing you couldn't teach me. But we need to figure out the answer for ourselves… I'm so happy I met everyone… I wish we could've gone on more adventures. But I guess we all have to say goodbye someday. Everyone… Thank you. Farewell. My memories will be part of the sky…

FF9

- Vivi

4.22 export Chinese via latex in org mode

org mode

C-c '

```
a = 2
b = 3
```

```
a + b
```