 理论上，python的确比C/C++慢（我对Java的开发没有经验，无法评论）。这一点不用质疑。 C/C++是编绎语言，直接使用的是机器指令，而python总是跑在的虚拟机上， 把一个逻辑直接转换为机器指令，和把同样的逻辑转换为字节码，然后再转换为机器码来执行相比， 总是要高上一大截，这一点，有各种各样的性能报告，我没有必要，也没有必要去做出这样一个报告来， 我记得大概的数值是一到两个数量级，用C++的话来说，这就是所谓的抽象惩罚。

        但实践上，并非这么简单。 现实开发者领域，总是新手远多于高手，而从新手转为高手的时间相比，C/C++使用者花费的时间要比Python使用者多的多，保守的估计是5~10倍。

        这有两方面原因，一方面， C/C++（尤其是C++）的确是“博大精深”，而且总是有无穷无尽的细节，如果你是一个真正的开发者，你就会明白，实际上，就完全精通C/C++编绎器的选项，就足够学两门python了（一点都不夸张，我遇到很多做两三年的C++开发者，遇到一个编译或者链接问题都一筹莫展），而编绎器仅仅是C++开发过程中的最后一环而已，你可以不懂，但你总得要花时间去弄懂，因为，作为一个开发者，这是每天都要面对的问题，否则你给别人一个什么东西，象python一样，给个源文件？：）等你搞懂了指针，搞懂了数组，你还得搞懂指针与数组之间微妙的关系，等你搞懂了引用，你又不得不搞清引用与指针的区别，等你弄清了static 不同的用法，你又得明白vector<vector<string>> 与vector<vector<string> > 是不一样的，尽管只是小小一个空格，然后呢？学习STL（不要告诉我你在C++直接不用STL开发应用），实际上STL对于现实应用来说，太差强人意了，如要你常做字符处理，你就明白我说的是什么了，然后Boost?然后...，终于有一天，你可以开始真正做应用了，一年过去了？：） 另一方面，C++的开发经历绝对是充满荆棘和惊喜，以及沮丧，等你达到C/C++的自由王国，你一定是身经百战，但脑袋中还是有一个弦在紧绷，因为，一不留神，你就又得没日没夜的与奇妙的错误作战，这就是C++的开发感受。而且，C/C++的编码风格问题，一直是开发中充满争议的问题，我见过太多的关于风格问题的，毫无意义的争吵，但是很极少见到最终哪一种风格会令人每个人信服。

       说说python吧，学习过程是这样的，数据类型是如此面向问题，没有了令人讨论的unsigned , short , long,这种这辈子我都不愿弄懂的修饰， 类型连同支持的操作是如此的易懂，一旦你明白了，从Sequence / Mapping纬上看问题，很容易区分list与dict，一旦你明白了Imutable与mutable的意义，你就很容易猜到，tuple与list ,set与frozenset, 的不同方法，一旦明白了list 的inplace-op与string的 non-inplace-op实现，就很容易猜到方法的使用方式，另一方面，符合现实问题的逻辑表达式，通用的len， 简洁的 a, b = b,a,让你觉得一切井然有序，而且简单明晰。强大而广泛且标准库，都把问题与实现的距离接近到了可以动手的距离。语法中相当部分约束，减少了不必要语言风格争论（私下认为，那怕是一个最难看的固定语法都比临时去决策如何去用来的好），这也加速了开发的步伐，况且还有一个风格标准建议在PEP中。

　　现实中，我不相信一个三个月或六个月的应用项目，使用python会比使用C++/C开发出来的系统慢。因为，开发者总是人， 拥有同样开发年限的C++/C程序员与python程序员对语言精通的程序是截然不同的，特别情况下，python程序员是已经真正进入了经验积累期，而可能C++/C程序员才刚刚有了开发的感觉。如果可以做这样一个实验，真实的情况，可能是这样的，python开发者，早就完成系统测试，着手进行性能的优化，考虑使用C/C++替换部分性能的瓶颈，而C/C++程序可能为集成测试的不完整在正在惴惴不安，信号是，由于程序员的技能问题，项目可能要延期。