**课程实践作业四**

将Guttag, John的《Introduction to Computation and Programming Using Python》中的：

6.2.3 When the Going Gets Tough

6.2.4 And When You Have Found “The” Bug

翻译为中文。

**提示：**

翻译重在“意达”，翻译是再创作的过程。

**要求：**

文档持续更新：

1. 推敲修饰翻译文字；
2. 结合课程学习，逐步更新加入自己编程的体验。

**提交：**

1. 文档名称：学号\_姓名\_作业四.docx；
2. 电邮：[cmh@seu.edu.cn](mailto:cmh@seu.edu.cn),

邮件主题：学号\_姓名\_作业四；

1. 推送到GITHUB。
2. 翻译

6.2.3 When the Going Gets Tough 当事情变得困难

Joseph P. Kennedy, father of President Kennedy, reputedly instructed his children, “When the going gets tough, the tough get going.”( 36 He also reputedly told JFK, “Don't buy a single vote more than necessary. I'll be damned if I'm going to pay for a landslide.”)But he never debugged a piece of software. This subsection contains a few pragmatic hints about what do when the debugging gets tough.

据说，肯尼迪总统的父亲约瑟夫·P·肯尼迪曾教育他的孩子们，“当事情变得艰难，坚强者继续前行（36 据说他也告诉过肯尼迪，“一张选票也没有必要买。如果我花钱取得了压倒性的胜利，我会被谴责”。）但是，他从来没有调试过一个程序。下面的段落给出了能帮助你在程序调试陷入困境时，找出问题所在的有效提示：

1. Look for the usual suspects. E.g., have you
2. Passed arguments to a function in the wrong order,
3. Misspelled a name, e.g., typed a lowercase letter when you should have typed an uppercase one,
4. Failed to reinitialize a variable,
5. Tested that two floating point values are equal (==) instead of nearly equal (remember that floating point arithmetic is not the same as the arithmetic you learned in school),
6. Tested for value equality (e.g., compared two lists by writing the expression L1 == L2) when you meant object equality (e.g., id(L1) == id(L2)),
7. Forgotten that some built-in function has a side effect,
8. Forgotten the () that turns a reference to an object of type function into a function invocation,
9. Created an unintentional alias, or
10. Made any other mistake that is typical for you.

寻找常见的错误，比如，你有没有

1. 将参数传给函数时顺序出错，
2. 拼写错误，例如，当你本该键入大写字母时，输入了小写字母，
3. 未能重新初始化变量，
4. 检查两个浮点数是否相等（==），而不是近似相等（记住浮点数的计算和你在学校所学的算术是不一样的），
5. 当你想让对象等价时（例如，id(L1)==id(L2)），检查它们的值是否相等（例如，通过表达式L1==L2来比较两列对象），
6. 忘记一些内置函数有副作用，
7. 忘记“()”会把对一个函数类型对象的引用变成对它的调用，
8. 创建了一个没有定义的对象，
9. 或者犯了其他你经常犯的错误。
10. Stop asking yourself why the program isn’t doing what you want it to. Instead, ask yourself why it is doing what it is. That should be an easier question to answer, and will probably be a good first step in figuring out how to fix the program.

别再问自己为什么程序没有像你想要的那样运行，而是问问自己为什么它像现在这样运行。这应该是个更容易回答的问题，并且可能是弄清楚如何修复这个程序的很好的第一步。

1. Keep in mind that the bug is probably not where you think it is. If it were, you would probably have found it long ago. One practical way to go about deciding where to look is asking where the bug cannot be. As Sherlock Holmes said, “Eliminate all other factors, and the one which remains must be the truth.” (37 ArthurConan Doyle, “The Sign of the Four.”)

记住，错误可能不在你以为的那个地方。如果它在你以为的地方，那你可能很早之前就找到它了。决定去哪里寻找错误的一个实用的方式是问问自己错误不可能出现在哪里。夏洛克·福尔摩斯说，“排除所有其他可能，剩下的那个一定是真相。”

1. Try to explain the problem to somebody else. We all develop blind spots. It is often the case that merely attempting to explain the problem to someone will lead you to see things you have missed. A good thing to try to explain is why the bug cannot be in certain places.

尝试着把这个问题解释给他人听。我们都有盲区。这种事经常发生，仅仅是试着向别人解释这个问题就能让你看到之前没注意到的。一个很好的方法是试着解释为什么错误没有出现在某个地方。

1. Don’t believe everything you read. In particular, don’t believe the documentation. The code may not be doing what the comments suggest.

不要相信你读到的全部东西，尤其是文献资料。代码可能不会按照意见zho那个建议的那样运行。

1. Stop debugging and start writing documentation. This will help you approach the problem from a different perspective.

停止调试程序而是开始记录问题。这将帮助你从一个不同的角度看问题。

1. Walk away, and try again tomorrow. This may mean that bug is fixed later in time than if you had stuck with it, but you will probably spend a lot less of your time looking for it. That is, it is possible to trade latency for efficiency. (Students, this is an excellent reason to start work on programming problem sets earlier rather than later!)

今天先到这儿明天再试。这可能意味着解决这个问题的时间会比你一直坚持下去晚一点，但你寻找这个问题所花的时间很可能会少很多。也就是说，用延迟来换取效率是有可能的。（同学们，这是一个很好的早一些而不是晚一点去开始做程序设计练习题的理由！）

6.2.4 And When You Have Found “The” Bug 当你已经找到问题时

When you think you have found a bug in your code, the temptation to start coding and testing a fix is almost irresistible. It is often better, however, to slow down a little. Remember that the goal is not to fix one bug, but to move rapidly and efficiently towards a bug-free program.

当你觉得你已经找到代码中的某个问题时，那种想要马上开始编程调试修改的冲动是几乎不可抵抗的。然而最好是放慢一点。记住最终目标不是修改一个错误，而是更快更有效率的向没有错误的程序前进。

Ask yourself if this bug explains all the observed symptoms, or whether it is just the tip of the iceberg. If the latter, it may be better to think about taking care of this bug in concert with other changes. Suppose, for example, that you have discovered that the bug is the result of having accidentally mutated a list. You could circumvent the problem locally (perhaps by making a copy of the list), or you could consider using a tuple instead of a list (since tuples are immutable), perhaps eliminating similar bugs elsewhere in the code.

问问你自己这个错误是否解释了所有已观察到的征兆，还是说它只是冰山一角。如果是后者，考虑修改这个错误时保持与其他变化相一致可能会更好。举个例子，设想一下你发现这个错误是不小心改变了一个列表导致的，你可以局部的避开这个问题（可能通过把列表备份），或者你可以考录用元组替代列表（因为元组是不变的），这样做说不定可以消除出现在程序其他地方类似的错误。

Before making any change, try and understand the ramification of the proposed “fix.” Will it break something else? Does it introduce excessive complexity? Does it offer the opportunity to tidy up other parts of the code?

在做任何更改之前，试着去了解推荐的“修改”的衍生问题。它会不会破坏其他地方？它会不会使程序过于复杂？它是不是能够为整理代码的其他部分提供机会？

Always make sure that you can get back to where you are. There is nothing more frustrating than realizing that a long series of changes have left you further from the goal than when you started, and having no way to get back to where you started. Disk space is usually plentiful. Use it to store old versions of your program.

始终确保你可以回到修改前的样子。没有什么比意识到很长一系列的改动使你远远偏离你开始时的目标，而又没有办法回到你开始的地方更令人沮丧。磁盘空间通常是很充裕的。用它来存储你旧版本的程序。

Finally, if there are many unexplained errors, you might consider whether finding and fixing bugs one at a time is even the right approach. Maybe you would be better off thinking about whether there is some better way to organize your program or some simpler algorithm that will be easier to implement correctly.

最后，如果有很多无法解释的错误，你可能会考虑一个一个找到错误并且修改是不是正确的做法。思考是否有更好的组织程序的办法或者一些能更轻松正确使用的简单算法，这样可能会使你的状况变好。

1. 体会

这两章主要提示了在发现程序出错并且不知道如何解决时的一些技巧。里面的内容并没有技术性很强，更多的是我们平时编程的时候也知道但是遇到问题就忽略的方法。其中有几个建议对我帮助很大。

1. 当你找的一个程序中的错误时，不能着急想要马上开始编程调试修改，而是放慢一点，仔细看一看上下文，这个错误是独立的，还是与前后有关联。仔细考虑过后再进行修改；
2. 遇到自己怎么也想不明白的问题时，不要钻牛角尖，尝试着把这个问题解释给他人听，或者利用网络寻求帮助，这样可以给思维上带来很大启发。
3. 随时记得要备份！