**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。

### 6.2.3When the Going Gets Tough

Joseph P. Kennedy, father of President Kennedy, reputedly instructed his children, “When the going gets tough, the tough get going.”36 But he never debugged a piece of software. This subsection contains a few pragmatic hints about what do when the debugging gets tough.

Kennedy教育他的孩子说“当处境变得艰难,顽强意志者会更努力去迎接挑战。”但他从未调试过一条程序。这件小事启示了我们当调试变得艰难的时候，我们该怎么做。

Look for the usual suspects. E.g., have you

* Passed arguments to a function in the wrong order,
* Misspelled a name, e.g., typed a lowercase letter when you should have typed an uppercase one,
* Failed to reinitialize a variable,
* 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),
* 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)),
* Forgotten that some built-in function has a side effect,
* Forgotten the () that turns a reference to an object of type function into a function invocation,
* Created an unintentional alias, or
* MadeMade any other mistake that is typical for you.
* 赋予函数命令行参数时顺序错误
* 拼错一个名字等等，当你应该输入一个大写字母的时候输入了一个小写字母
* 重新定义一个变量时出错
* 检验两个浮点数是否相等（==）而不是近似相等（记住：浮点数的四则运算与你在学校学的不一样）
* 检验数值相等(L1==L2指的是语句相等id(L1)==id(L2)指的是对象相等)。
* 忘记一些内置函数有副作用
* 忘记()把引用转化为函数对的对象参数的调用
* 创造了一个无意义的别名，或者制造任何对你来说典型的错
* 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.

停止问你自己为什么项目没有像你想的那样做。相反，问你自己为什么它在做，它在做什么。那将会是一个更容易回答的问题，并且可能会帮助你弄清楚怎么解决问题。

* 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.”

记住错误可能不在你认为的地方。如果它在，你很久前就已经找到了。一个可行的办法是去找不可能错的。如夏洛克所说，“排除所有的因素，剩下的就是事实”

* 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.

试着去向别人解释问题。我们所有人都在扫除盲点。通常是向别人稍作解释你将会看到你错过的东西。尝试着解释为什么错误不可能出现在这些地方是一件好事

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

不要相信你看过的一切。特别是，不要相信文献。那些评论建议不能帮助你写代码。

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

停止调试并且开始写文档，这将会帮助你从个人层面解决问题。

* 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!

走开，明天再试。这可能意味着你能够迅速找到问题，但是解决不了而卡在那里。这时，可能就是用时间换效率。（学生，这就是为什么着手解决程序问题赶早不赶晚）

**6.2.4 And When You Have Found “The” Bug**

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.

当你认为你已经找到你代码中的错误，想要开始写代码和调试错误的冲动常常是不可抵抗的。但是，慢一点常常会更好。记住你的目的不是消除错误，而是迅速有效地做好没有错误的程序。

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.

问问你自己是否这个错误解释了所有可观察到的问题症状，或者是否只是冰山一角。如果是后者，最好考虑把相似问题归类以区别于其他变化。假如，你已经发现了那个bug导致意外改变一串列表。你应该绕开当地问题（也许通过拷贝一串列表），或者你可以考虑用一组复表代替一串列表（因为复表是不可变的），也许能够排除在代码中其他地方相似的错误。

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?

在做任何改变之前，试着弄清楚被提议的修改的的衍生影响。它会不会破坏别的？它会不会导致过于复杂化？它会不会有利于代码其他部分的整理？

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.

总要确保你可以回到未修改之前。没有什么比这件事更令人沮丧的了，那就是意识到一系列的修改反而让你背离目标，而且你无法回到你开始的地方。磁盘空间通常是很充裕的，用它储存你之前旧的程序版本。

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，你可能要考虑是不是一次性找到并修改错误才是更正确的办法。也许你要思考是不是有更好的组织程序的办法，或者一些更简单的算法使得程序的正确实施更容易。