**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。

**原文：**

6.2.3 When 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 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.

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

o Passed arguments to a function in the wrong order,

o Misspelled a name, e.g., typed a lowercase letter when you should have typed an uppercase one,

o Failed to reinitialize a variable,

o 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),

o 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)),

o Forgotten that some built-in function has a side effect,

o Forgotten the () that turns a reference to an object of type function into a function invocation,

o Created an unintentional alias, or

o Made any other mistake that is typical for you.

• 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.

• 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.”)

• 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.

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

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

• 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!)

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.

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.

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?

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.

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.

**翻译：**

6.2.3当事情开始变得困难

肯尼迪总统的父亲曾近教育他的孩子们说：“当事情开始变得困难的时候，越是坚强的人越是容易越过这些困哪。”（据说他还告诉肯尼迪说：“千万不要用金钱购买不必须的选票，如果用钱获得了这个，我会因此而谴责自己。”）虽然他并没有调试过任何一个程序，但这确是共通的。在调试程序中遇到困难时，之后的段落会给你一些切实可行的提示：

寻找那些常犯的错误，例如：

1.将参数传递给函数时搞错了原有的顺序；

2.拼错了函数名，例如将小写的字母输成了大写；

3.重新初始化变量失败；

4.测试两个浮点型数值时用了完全相等而非近似相等（记住这一点：浮点数运算时和你之前在学校中学到的计算方法并不完全相等）；

5.当你的意指两对象的值相等时（例如id（L1）==id（L2）），却去检查它们的数值是否相等了（例如：通过写出两列表达式L1==L2来比较了）；

6.忘记了一些内置函数的副作用；

7.忘记括号的使用“（）”会将一个队函数对象的引用变为函数的调用；

8.平白创建了一个定义的对象的别名；

9.或是犯了一些你常犯的典型错误。

不要一再的问为什么程序没有按照你原本设想的那样运行。相反，应该问你自己为什么程序像现在这样运行的。这会是一个更简单回答的问题，并且可能是解决现有程序问题的良好开端。

记住bug很有可能不在你认为它在的地方。如果它在那儿的话，你很有可能在很久以前就已经找到了它。一种切实可行的方法是问自己bug不可能在哪里。就像夏洛克福尔摩斯说的；“排除所有的可能性，那么剩下的一定是真相。”

尝试着把问题告诉别人。我们生来就有盲点。经常会有这种情况当我们尝试向他人阐述自己发现的问题时却让我们找到了我们未曾看见的缺漏。去解释为什么错误没发生在你想它在的地方可能会是一个很好的方法。

不要相信你曾经从书中读到的一切，尤其是那些文献资料。代码可能没有按照它所建议的方向运行。

停止调试程序并将遇到的错误写下来。这将帮你从不同的方面找到解决问的方法。

忽略后明天继续尝试。这意味着解决这个错误的时间会比你一直坚持下去要长，但你花在这个问题上总的时间可能会少的多。也就是说我们能用等待来换取效率。（同学们，这是个极好的理由让我们早一些而并非晚一些去研究编程中的问题。）

6.2.4当你找到bug后

当你认为你已经找到了代码中的一个错误，那种想要开始测试并改正这个错误的想法是不可抗拒的。但通常，放慢一点可能会更好。记住一件事，那就是我们的目的并不是去修改一个错误而是快速高效的完成一个没有错误的程序。

尝试问问自己这个错误能否解释所有的可观察到的结果，或者它只不过是冰山一角。如果是后者，最好要考虑在修改的同时保持和其它变化一致。假想这么一个例子，你已经发现错误来自于不当心的改变了一个列表，你可以局部的避开这个问题（也许是将列表备份），或者用元组代替列表，这样也能消除代码中的其它部分出现相同的错误。

在做出修改之前，尝试理解这修正是否会引发其它的问题，是否会变得过于复杂，是否会要修改代码的其它部分。

一定要确保能够恢复到之前的样子。没有什么比意识到大量的修改之后使你离目标越来越远并且没办法回到一开始的样子更令人沮丧。硬盘空间一般是很大的，可以用于存放旧版本的代码。

最后，如果有许多没法解释的错误，你也许应该思一下是否一个个找到错误然后修改是不是一个好的解决办法。也许你应该想想其它的程序组织方式或是一些更简单地方式算法，这样可能让问题变得简单。

**小结：**

程序设计不是一蹴而就的，是一个循序渐进的过程，在遇到问题时，不要急于去着手修改自己的程序，先静下来好好地想一想，尽量更全面的看清问题的所在，不要急于解决问题，重要的不是问题本身，而是我们如何更快速高效的解决问题，问题不可怕，可怕的是被问题吓住，像文中所说的好好审视问题，才是解决问题最有效的办法。