**课程实践作业四**

**题目要求**

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。

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。

1. **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.

1. **译文**

6.2.3 当情况变得艰苦时

肯尼迪总统的父亲，Joseph P. Kennedy在教育他的孩子们时说：“当事情变得艰难的时候，意志坚定的人将保持前进。”即时他从来没有调试过任何软件。在你调试程序陷入困境时，下面的段落将给你一些如何去做的实际的提升。

寻找通常易犯的错误，例如：你有没有

1.将参数传递给函数时弄错了顺序。

2.拼错一个名字，例如：当你应该输入一个大写字母时你却输入了小写字母。

3.重新初始化变量失败。

4.测试两个浮点值是否相等(= =)而不是几乎相等(请记住,浮点运算和你在学校中所学到的算法不一样)。

5.当你想要检测两个对象是否相等时（例如：id（L1）==id（L2）），却检测它们的数值是否相等（例如：通过表达式L1==L2来比较两列数据）。

6.忘记一些内置函数的副作用。

7.忘记“()”会将一个对函数类型对象的引用转变成对函数的调用。

8.无意中创建对象的别名。

9.犯一些你经常会犯的错误。

别再问自己程序为什么没有像自己想要的那样运行。相反的，你应该问自己程序为什么会像现在这样。那将会是一个必较容易回答的问题。并且这将会是很好的第一步去找出如何修复程序。

请记住，错误可能出现在不是你认为的地方。如果在，那么你很久之前就能找到它。决定从何处开始找错误最实用的方法就是问自己错误最不可能出现在哪里。就像福尔摩斯说的：“排除其他所有可能性，剩下的就是真相。”

尝试将你遇到的错误说明给别人。我们每个人都有盲点，往往在向别人说明你遇到的问题时，会发现你错过的东西。一个最好的方法就是去解释为什么错误没有出现在预料的地方。

不要相信所有你读的东西，尤其是那些文献资料。你的程序并不一定会像解释的那样运行。

停止调试并开始记录问题，这将帮你从不同角度来找解决问题的方法。

停下来走开，明天在尝试。比起你一直坚持做下去，这可能意味着会更晚解决问题，但是你可能会花更少的时间来寻找问题的所在。就是说，我们可以用等待来换取效率（同学们，这是一个很好的早一些而不是更晚去开始课程练习题程序设计的理由！）。

6.2.4 当你找到错误的时候

当你认为你已经找到了程序中的错误时，那种想要马上开始编写并测试修改程序的冲动是难以抗拒的。然而放慢一点节奏是更好的选择。记住我们的目标并不是去修改一个错误，而是要更快更有效率的向着正确的程序前进。

问问自己这个错误是否能够解释目前观察到的所有症状，异或者只是其中的冰山一角。如果是后者，那么你最好要考虑在修改这个错误的时候，保持它和其它的变化相一致。举个例子，假如你已经发现错误的原因是偶然改变了一个列表，你可以局部的避开这个问题(也行是通过将这个列表备份) ，或者可以考虑用元组来替代列表(因为元组是保持不变的)，这样也行能够消除在代码中其他部分出现相同错误。

在做任何改变之前，尝试并且认识到这个改变可能造成的其他错误，这样的改变会破坏其他部分么？它会让程序变得异常复杂么？ 它是否能够为你提供更好的机会去整理程序的其他部分？

总是确保你能够回到修改之前。没有什么比意识到自己做的一长串修改使你离目标越来越远时，却没有办法再次回到你最开始的地方时更加沮丧。磁盘的空间往往是很充足的，所以记得用它来储存你的旧版本程序。

最后，如果出现了许多难以解释的错误时，你应该思考一下逐一找到错误并且修改他们是不是正确的方法。也许你应该思考是否有更好的方法来组织你的程序或者一些简单的算法，这样将更加容易让程序实现正确。