**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。

**一、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.” 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.”

• 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.

**二、译文**

**6.2.3当事情变得棘手**

据说约瑟夫·P·肯尼迪，肯尼迪总统的父亲，曾教导自己的孩子：“当事情变得棘手的时候，棘手之事就变得简单。”但是他从未调试过一款软件。这段话包含着一些当软件调试变得困难时候的实际性提示。

·找寻最寻常的怀疑。例如，你曾经

o 用一个错误的顺序将参数传递给了函数么？

o拼错一个名字，例如当你该输一个大写字母的时候输了小写字母？

o重置一个变量时未能成功？

o测试得到两个浮点值是相等的（恒等于==）而不是近似等于（记住，浮点算法跟你在学校学到的一般算法是不一样的）？

o当你想着证明对象相等（例如id(L1)==id(L2)）的时候证明了值相等（例如通过写表达式L1==L2来比较两组列表）？

o忘记一些内置功能有副作用？

o忘了将引用一个对象类型的函数变成函数调用的（）？

o无意中创造了新名词或者……？

o犯了一些对你来说很典型的错误？

·不要再问自己为什么程序没有按照预想的那样运行，相反地，问问自己为啥它会这样运行。那个问题将会很好回答，并且极有可能会是找出如何完善程序一个良好的开端。

·请记住错误极有可能不在你想到的那个位置。如果真是这样的话，你早就该找到它了。决定到哪找寻错误的一个实际的方法就是找找错误在哪不可能存在。正如福尔摩斯所说，“消除别的因素所在，依然存在的将会是事实。”

·尝试着将这个问题解释给别人听，我们都会发现盲点。通常是这种情况，试图给别人解释问题之所在时将会引导你发现你错过了的事情。试着去解释的很好的事情就是为何问题会出现在这样的位置。

·你所读到的东西不要全都相信，特别地，不要相信文档。代码可能没有像评论所显示的那样工作。

·停止调试并且开始写文稿，这会帮助你从另一个层面接近问题的本质。

·走开，明天再试。这或许就意味着如果你被它卡住了，那个问题在时间上将会晚一些被修复，但是你将极有可能会话少很多的时间来找寻这个错误。也就是说，极有可能存在着效率延时。（对学生来说，这是一个很好的理由来更早地而不是更晚地从事编程的问题设置。）

**6.2.4当你发现问题之所在**

当你觉得在自己的代码中发现了错误时，开始编码和测试修正的诱惑几乎是不可抗拒的。然而，稍慢一点通常会更好一点。谨记在心，最终目标不是去修正一个错误而是朝着一个不存在错误的程序方向高速有效地运行。

问一问自己这段代码是否阐释了所有可能观察到的情况或者说只是冰山一角。如果是后者的话，考虑下与这个问题相一致的其他变化或许会更好。例如，假设你发现这个问题是无意中改变列表所得到的，你可能会很自然地规避这个问题（也许是对之前的列表做一个备份），或者你可能会考虑用一个元组来代替这个列表（因为元组是不可变的），或者消除代码中别的地方与之相似的错误。

在做任何改变之前，尝试着理解已经提出了的“修正”的分支。它将破坏别的一些东西么？它介绍过度复杂性么？它提供整理好代码别的部分的机会吗？

常常确认下你能返回到你在的地方。没有比你开始后发现一系列的变化使得你离刚开始原来越远或者无法回到最初的状态更令人沮丧的了。磁盘空间往往是充足的，用它来储存下你原来的程序。

最后，如果有许多无法解释的错误，你或许该考虑考虑一次发现并修正一个错误是否是正确的途径。或许你该考虑下是否有更好的方式来组织你的程序或者简单点的易于正确实现的算法。

**三、小结**

对于像Python这样的软件的学习，多读一读国外先进的教材是很有必要的，特别是在市场上没有很好的译本的时候，这也就需要我们能够自己很好地来对原书作翻译与注释。在对这两部分进行翻译的过程中，英语水平这一点就不再多说，主要还是对于一些专业术语的翻译欠完善，很多还是借助于有道词典进行翻译的，虽说如此，但还是理解了文段的大概含义，对于软件编程也有了一个更好的认识。相信在后面对Python的接触与学习中，我会掌握更多更完备的知识，争取做到尽善尽美。