**《Introduction to Computation and Programming Using Python》翻译作业**

**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 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肯尼迪是这样教导他的孩子们的，“麻烦越大，斗志越强。”但是他从来没有调试过一丁点程序。本部分包含了一些当调试变得困难的时候的实用提示。

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

寻找常见的疑问，例如你曾遇到过以下情况吗

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给函数传参时顺序错误

o拼错了单词。例如，在本应该输入大写字母的地方输入了小写字母

o重新初始化变量失败

o测试两个浮点数完全相等（==）而不是近似相等（记住浮点算法和你在学校学到过的算术是不一样的）

o想测试对象相等（例如，id(L1)==id(L2)）却测成了值相等（例如，通过语句L1==L2来比较两个数组）

o忘记了一些内置函数有副作用

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.

o 忘记（）会将类型函数的对象的引用转变成函数调用。

o不小心创建了别名

o犯了一些其他的你常犯的错误

*•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.

不要再问你自己为什么程序不按你设想的运行。而是应该问你自己为什么它会这样运行。这应该是一个更容易回答的问题，并且可能会成为解决程序问题的良好开端。

*•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，bug可能并不在你认为的地方。如果是在那里，那可能你很久以前就发现了。一个有用的寻找问题在哪里的方法是问自己bug不可能在哪里。正如夏洛克福尔摩斯所说“排除所有不可能因素，剩下的那个就是真相”

*•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.

尝试着向其他人解释这个问题。我们都有盲点。事实是只是尝试着向其他人解释这个问题就会让你看到你错过的事情。你可以尝试着解释为什么bug不可能在那个地方。

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

不要相信你读到的一切。特别是不要过于相信文档。代码可能并不能像注释所说的那样运行。

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

停止调试并且开始书写文档。这会帮助你从不同的角度解决问题。

*•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。这可能意味着修复bug在时间上会比你一直钻研在上面要晚，但你可能只要花更少的时间来寻找bug。那就是说用延迟来换效率是有可能的。（同学们，这说明早点开始程序方面的工作比拖到很晚要好得多！）

**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.

当你认为你已经发现了你代码中的bug，开始写代码和调试的诱惑几乎是不可抵挡的。但是稍微慢一点往往会更好。记住目标不是修复一个bug，而是朝着一个没有bug的程序快速高效的前进。

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bug是否解释了所有可见的症状，或者它是否只是冰山一角。如果是后者，可能和其他的变化一起思考来解决这个问题会更好。举个例子，假定你已经发现bug是由于意外地改变了列表。你可以在本地避开这个问题（可能通过复制这个列表），或者你可以考虑使用元组而不是数组（因为元组是不可改变的），可能可以消除代码中其他地方的相似的bug。

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?

在做任何变动前，尝试并且理解修复提出的后果。它会导致其他问题吗？它会导致过于复杂吗？它会帮助整理其他部分的代码吗？

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.

你要确信你总能返回到你原来的位置。没有什么事情比意识到一系列的改变令你比最开始的时候离自己的目标更远，并且还无法回到开始的地方更令人沮丧了。磁盘空间通常是很充足的。用它来存放你程序的旧版本。

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，你可能要思考一次发现并修复一个bug是否是正确的途径。可能考虑是否有其他更好的方法来组织你的程序或者是否有正确完成程序的更容易的简单算法会更好。