《Introduction to Computation and Programming Using Python》部分翻译

1. 英文原文

**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 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

*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、翻译

6.2.3 当黑暗笼罩

肯尼迪总统的父亲，Jodesph P. Kennedy，据说常常如是教导他的子女：“当黑暗笼罩，则黎明不远。” 但他从来没有调试过哪怕一点点的软件。本小节包含了一些当你的调试陷入窘境时该如何做的实用小提示。

－去找找通常会遇到的问题， 比如， 你曾否

＊将变量以错误的顺序传给了函数

＊拼错了名字，比如，将大写打成了小写

＊没有重新初始化一个变量

＊尝试用等于号（＝＝）去比较两个浮点数，而不是近似的相等（记住浮点数运算和你在学校学的不一样）此处应该用abs(a-b)<0.00001这样的方式比较

＊试一下数值等式（比如：用L1==L2比较两列）当你想表达对象相等比较时（如：id(L1)==id(L2)）

＊忘了某些内置函数有副作用

＊忘了()会使得一个type函数对象的引用变为函数调用

＊创建了一个无意义的别名

＊犯了你常犯的典型错误

－不要问你自己为什么你的程序没有做你期望它做的事，而是问你自己为什么它会做它现在在做的事。这样应该更容易回答，而且很可能是你迈向解决问题的第一步。

－永远记住，bug很可能不在你认为它在的地方，要是在的话，你早就该找到了。一个找bug在哪里的很实用的方式就是问一下bug不会在哪里。正如福尔摩斯所说的，“排除所有其他因素，剩下的一定是真相。

－尝试向别人解释一下你的问题。我们的有自己的盲点，经常我们能在向别人解释的时候发现自己错过的点。一个尝试向别人解释的好的点是：为什么bug不能在某一个的地方。

－ 不要相信你读到的任何东西。特别来说，不要相信文档。别人的代码很可能做的并不是他的注释所写的那样。

－ 停止调试并开始写文档。这回帮助你从另一个角度接近问题。

－走人，明天再来试试。这可能意味着你的bug会比你死磕这个bug的时候要晚一些被调好，但是你可能会花费少很多的时间去寻找它。也就是说，这是有可能靠延迟来换取效率的。（学生们，这也是一个很好的理由让你们越早开始编程越好）

6.2.4 当你找到了那个bug

当你认为你找到了你代码中的那个bug时，开始码代码并测试你的修改的诱惑几乎是不可拒绝的。然而这个时候稍微放慢一点往往更好。记住你的目标不是修复一个bug，而是快速且高效的完成一个全对的程序。

扪心自问这个bug是否能解释所有观察到的错点，还是说它只是冰山一角。如果是后者，那么在小心的修复这个bug的时候同时注意它所带来的改变将更有裨益。比如，假设你已经发现这个bug是使得一个列表变异的罪魁祸首。你应当首先在本地设法绕过这个问题（比如做一份正确列表的拷贝），或者你应当考虑使用一个元组而非列表（由于元组是不可变的），可能消除这个代码中其他同样的bug。

在作出任何的尝试下，试着理解你提出的“修复方案”的后果。这样会不会造成其他不良后果？它是否加入了过多的复杂度？它是否提供了整理其他部分的机会？ 总是确保你可以回到你原来所在的地方。没有什么比长时间调试后发现自己比开始时离目标更远了，而且还回不去更让人沮丧的了。硬盘空间通常是足够的，用它存储你旧版本的程序。

最后，如果有其他无法解释的错误，你可能要考虑一下你找的和解决的bug是不是对的方向。可能你停下来想想有没有更好的组织你程序的方式或者一些能正确实现你目标的更简单的算法会更好。

1. 心得与体会

接下来讲一讲我在调试过程中的体会。首先，python当中浮点数的比较这一点是很值得注意的，在利用python计算作业的过程中就有这样的问题。因此我特别标记了红色，这也是我找到的错误之一。由于现在暂时用python编程次数并不多，但是这几条debug技巧我还是颇有心得的。在大二第一学期做MFC工程时，我做了一个打字游戏。这个工程文件比较多，代码包括了自动生成的和手打的大概有五六百行。有一个问题就是每次更新之后，单词下落速度会增加。一开始我以为是循环出错了，便改了循环。然而并没有变化，最后发现是循环没有问题，但由于数组忘记清空导致了循环所在函数会被调用多次。这就属于上述问题当中，急匆匆找到一个错误就立刻下手去改。

总结上述经验，我认为有一句话很重要，但作者没有说。他说要考虑一下为什么代码结果会是这样，我想更应该用log或者输出检查一下你的思考，再开始动手改动代码。这是较大工程时一定要注意的。

之后若python利用中，再有心得，将会继续加在此处。