**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：

1. 翻译重在“意达”，翻译中能体现自己理解的“意”为佳；
2. 翻译以后根据自己编程体验，写体会。
3. 翻译文档内容：英文原文、翻译和编程体验
4. **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.” (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**

据说，肯尼迪总统的父亲约瑟夫·P·肯尼迪曾教育他的孩子们：“当情况变得艰苦时，意志坚定的人却能够继续前行。”即使他不曾运行调试过任何软件。下面的段落将在你进行程序调试陷入困境时，提供一些解决问题的有效方法。

寻找那些常见的易犯的错误，例如：你是否已经

1. 将参数传递给函数时弄错了顺序；
2. 拼错名字，比如：在你需要输入大写字母时，却输成了小写；
3. 重载变量失败；
4. 测试两个浮点类型数据是否完全相等而不是约等（记住，浮点运算和你在学校里所学的并不完全相同）；
5. 在你想要检测两个对象是否相等时（例如：id（L1）==id（L2）），却做成了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. 忘记一些内置函数是有副作用的；
7. 忘记“（）”会将一个对函数类型对象的引用转变成函数的调用；
8. 创建一个没有定义过的对象；
9. 犯一些自己常犯的错误；

别再追问自己为什么程序没有像你所期待的那样运行。相反，你应该问问自己为什么程序会像现在这样。那会是一个更加容易回答的问题。并且，这将会是你在完善程序时所迈出的最好的第一步。

记住错误也可能并没有出现在你认为它应该出现的地方。如果它真的出现在了那儿，那么你应该早就找到了它。一个决定从哪里开始寻找错误的最有实用性的方法是问自己错误最不可能出现在哪里。就像福尔摩斯所说，“排除其它所有的可能性，剩下的那个一定就是真相。”

尝试着向别人阐述你所遇到的问题。我们每个人都有盲点，很多时候在向别人阐述自己的问题的时候可能会帮助你发现自己所遗漏的东西。一个最好的方法是尝试着去解释为什么错误没有发生在你所认为应该出现的地方。

不要相信你所读过的任何东西，尤其是那些文献资料。你的程序也许并没有按照意见中所说的那样运行。

停止调试并开始写下其中的一些问题。这会帮助你从另一个角度寻找到解决问题的方法。

先暂停手中的程序，过些时间再来尝试编写。这可能意味着解决这个错误的时间会比你一直坚持下去要来的晚，但是你可能会花更少的时间来寻找错误。也就是说，我们也许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 当你找到了问题**

当你认为你已经找到了程序中的错误时，那种想要马上就开始编写并测试修改的冲动是难以抗拒的。但是，往往放慢一些是更好的选择。要记住，我们的目标并不是去修正一个错误，而是更快而且更有效率地向着没有错误的程序前进。

问问自己这个错误是否能解释所有可观察到的症状，或者它是否只是冰山一角。如果是后者，那么你最好在考虑修改这个错误的时候要保持它和其它的变化相一致。举个例子，假设你已经发现错误来自于不小心改变了一个列表，你可以局部地回避这个问题（也许是通过将列表进行备份），或者可以考虑用元组来替代列表（因为元组是保持不变的），这样也许能够消除代码中其它部分的相似错误。

在做任何改变之前，尝试着去思考这个修改可能衍生出的其他问题。它会使程序变得太过复杂吗？它是否能为你整理程序的其它部分创造更好的机会呢？

你要永远确保自己能够回到修改之前。没有什么比在意识到自己做的一长串的修改反而使你离目标越来越远并且没有办法能再次回到你最开始的地方更加让人沮丧。磁盘的空间通常都很充足，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，你也许应该思考一下逐个找到错误并修改是不是正确的方法，思考一下是否有更好的解决问题的方法或者一些能更加轻松正确使用的算法，这样也许会使你的情况变得更好一些。

**个人体会与总结：**

由于基本上还没有用Python语言编写过程序，这里就简单的从学习C++语言上谈论一下个人体会。上面列出了一系列我们编程时出现的问题，个人觉得总结的很到位。也许是自己在计算机语言这方面没有什么天分吧，大一的时候我就对C++感到莫名的恐惧，然后一直延续到其他的计算机语言上。虽然学习了一年的C++语言，但到了现在却对它理解的还是远远不够。不过据以往的经验来说，想要熟识一门语言（是的，软件工程老师在开学时就讲过，不要试图去掌握一门语言），就得大量的实践操作。现在对于软件工程这门课，我想也是一样的吧，熟能生巧，自己多去琢磨、多去尝试，总能不断总结经验越做越好。

对于这两段小文章的翻译，有些地方还是有一些难度的，其中有一些专业词汇不认识，通过参考其他同学的翻译以及查词典，大概能够弄懂个中含义吧。翻译得不是很好，这大概是我自己能够理解的一些意思。