**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**

下文中红字为翻译时觉得有点难度的地方，翻译到一半才开始标注，因此标注不是很全。

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当事情变得艰难

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，“当事情变得艰难，意志坚定的人还能继续前进。”（注释36：据说他还告诉肯尼迪，“一张多余的选票也不要买。如果我用钱取得了压倒性的胜利，我会被定罪。”）但他从未调试一个软件。下面的段落给出了能帮助你在程序调试陷入困境时，找出问题所在的有效提示。

1. 寻找常见的错误。例如，你有没有：
   * 1. 将参数传递给函数的时候顺序出错；
     2. 拼错一个名称，例如，当你应该键入一个大写的字母时，输入了小写字母；
     3. 未能重新初始化变量；
     4. 检查两个浮点值是否相等（==），而不是近似相等（记住，浮点运算和你在学校所学的算术是不一样的）；
     5. 在你表示对象的等价关系时（例如，id(L1)==id(L2)），检查它们的值是否相等（例如，通过写表达式L1==L2比较两列对象）；
     6. 忘记一些内置的函数有副作用；
     7. 忘记“()”会把对一个函数类型对象的引用变成对它的调用；
     8. 创建了一个没有定义的对象；
     9. 或犯了一些其他的你经常犯的典型错误。
2. 停止问自己为什么程序不是按照你所想的在运行，相反，问问你自己为什么程序像现在一样运行。这应该是一个比较容易回答的问题，而且可能是搞清楚如何解决这个问题的很好的第一步。
3. 请记住，错误可能不在你所认为它在的地方。如果它在你所认为的地方，你可能很久以前就发现它了。决定去哪里寻找错误的一个实用的方式是问问自己错误不可能出现在哪里。夏洛克·福尔摩斯说，“排除所有其他可能，剩下的那个一定是真相。”（注释37：阿瑟·柯南·道尔，“四签名”。）
4. 试着将这个问题解释给其他人。我们都有盲点。往往是这样，仅仅试图向他人解释这个问题的就可以使你看到你之前没有注意到的东西。一个很好的方法是试着解释为什么错误没有出现在某个地方。
5. 不要相信你读到的任何东西，尤其是，不要相信那些文献资料。你的代码也许并没有按照意见中所说的那样运行。
6. 停止调试并开始写下其中的一些问题。这将帮助你从不同的角度找到解决问题的方法。
7. 离开，明天再继续尝试。这可能意味着在时间上解决这个错误会比你一直坚持下去要晚一些，但是你可能会花更少的时间来寻找错误。也就是说，可以用延迟来换取效率。（同学们，这是一个很好的早一些而不是晚一点去开始做程序设计练习题的理由！）

6.2.4当你已经找到错误

当你认为你发现了你代码中的一个错误时，那种使你开始编码和调试修改的诱惑几乎是不可抗拒的。然而，放慢一点往往更好。请记住，我们的目标不是为了修复一个错误，而是要快速高效地向一个没有错误的程序前进。

问问自己，这个错误是否解释了所有观察到的症状，或者它是否只是冰山的一角。如果是后者，考虑修改这个错误时保持与其他变化相一致可能会更好。假定这样一个例子，你发现错误是由于不小心改变了一个列表导致的。你可以（也许是通过将列表备份）局部地绕过这个问题，或者你可以考虑用元组替代列表（因为元组是不可变的），这样也许就消除了代码中其他地方类似的错误。

作出任何改变前，尽量了解建议 “修改”的衍生问题。这个修改将会破坏其他地方吗？是否会导致程序过于复杂？它是不是能够为整理代码的其他部分提供机会？

始终确保你可以回到修改前的样子。没有什么比意识到通过很长一系列的改动你远远偏离你开始时的目标，而又没有办法回到你开始的地方更令人沮丧。磁盘空间通常是很充裕的。用它来存储你旧版本的程序。

最后，如​​果有许多无法解释的错误，你可能会思考是否一个一个的查找并修复错误还是正确的做法。也许你考虑一下是否有更好的方式组织你的程序或者一些能够使程序更容易正确实现的简单算法，你的境况会更好。

1. **心得体会**

由于还没有用Python语言编写很多程序，只做过几次简单尝试，因此体会不是很深刻，编程方面的体会只能结合所学过的C++语言写一写，不过这两种计算机语言的编程也有不少相似的地方，倒也并无大碍。6.2.3中列出了一些常见的编程错误，很多都是由于我们编程时的粗心大意造成的，除此之外，还会遇见一些令我们百思不得其解的错误，排查错误的过程需要我们在头脑冷静时进行，我们往往想当然的认为某些地方绝对不会出错或者卡在某个错误上无法解决。经过各种尝试后我们不妨换个解决问题的方法，将问题暂时搁置，是自己混乱的大脑清醒一下，也可以顺便找其他人问一问探讨一下，也许聊着聊着或者突然灵光一现就发现了自己错误的解决方法。

在对这两小节文章进行翻译时遇到了不少困难，收获和感触可谓颇多。

困难主要源于一些专业词汇和表达方法以及一些长长的句子。文章中有一些是完全没见过的生词，通过一些词典可以轻松解决，倒是有一些常用的词结果放在句中好像并不是经常用的那个意思，还好以前学过C++，可以大致推断出符合编程语境的意思。然而并不是知道了每个单词的意思就能够将一段话翻译好，还要联系上下文，使前后语句读起来比较流畅，加上中英表达各种顺序的差异，虽然只有八百个单词，倒也耗费了不少时间。

收获方面除了学到了不少专业性的词汇与表达方式，锻炼了一下英语翻译能力外，最重要的是了解了一些如何找出程序错误的方法。另外，start work on programming problem sets earlier rather than later这句令我印象很深，老师上课时就说你们现在就可以做作业八了，当时我还觉得是不是早了点，看完这几段文章发现越早开始程序的编写越好的确很有道理，这样能够尽早的发现问题，有更多的时间去解决问题，而不是临近课程结束了再匆匆编写。