**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。

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

**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*
* *Passed arguments to a function in the wrong order,*
* *Misspelled a name, e.g., typed a lowercase letter when you should have typed an uppercase one,*
* *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),*
* *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)),*
* *Forgotten that some built-in function has a side effect,*
* *Forgotten the () that turns a reference to an object of type function into a function invocation,*
* *Created an unintentional alias, or*
* *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.

6.2.3 举步维艰之时

约瑟夫·P·肯尼迪，肯尼迪总统的父亲，据说教育他的孩子们，“当事情变得艰难,坚强者才能继续行路 。”但他从未调试一个软件。此小节包含有关调试时遇到困难时实用的应对方法。

* *寻找常见的错误，例如，你有没有*
* *将参数传递给函数时顺序错误*
* *拼写错误，例如：大小写错误*
* *变量初始化失败*
* *检查两个浮点类型变量是否相等而不是约等于（浮点数运算和你在学校里学习的运算是不一样的）*
* *当你想要两个对象相等时，检查他们的数值是否相等（检验语句L1==L2）*
* *忽略了一些内置函数的副作用*
* *忽略（）将函数类型对象的引用转换成函数调用的作用*
* *使用了一个未定义的函数或变量的别名*
* *其他常犯错误*
* *别问自己为什么程序是不是按你要求的运行。相反，问自己，为什么它这样运行。这相对于前者更加容易，并且很可能成为你明白如何改正程序的第一步。*
* *牢记错误可能会出现在你想象不到的位置。如果它是这样，你可能需要很久来发现它。一个实用的方法是去检查你觉得错误不可能出现的位置。如夏洛克福尔摩斯所说，“抛开所有不可能的,剩下的,不管多么匪夷所思,那都是事实。”*
* *尝试向其他人解释这个问题。我们都有思维盲点。当你将自己所遇到的问题阐述给他人时你也会发现你所遗漏的一些细节。一个好的例子比如说你可以解释为什么错误没有发生在你所预定的位置这一问题。*
* *不要相信你阅读的任何东西。尤其是，不要相信文档。代码可能不会像批注解释的那样运行。*
* *停止调试程序，记录下你的编程过程。这也许有助于你从其他角度发现问题。*
* *放下工作，明天再继续。这可能意味着这个问题相比于你坚持做完会更晚解决，但是相对来说你可能会花更少的时间来寻找它，也就是说提高了你的工作效率。（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 当你找到错误后

当你发现错误之后，不要急于着手解决问题，你的目的并不只在于解决程序中的一个错误，而是为了以后编写程序不出现类似的错误。

首先，思考这个错误是否是整个程序出错的根源所在或只是众多错误中的冰山一角，若是后者，调试时要注意此处错误与其他变化的一致性，例如当你发现了一个错误的原因是改变了一个列表，你可以局部地避开这一问题，或者可以用元组代替列表，这样做或许会消除程序中其他部位的类似错误。

更改程序前，尝试去思考各种不同的修改方法会不会带来其他的问题，它会不会使程序的其他地方出错或使程序过于冗长？是否能够使程序的其他部分变得简洁？

确保你可以使你的程序恢复到修改以前，以防你费尽心思修改你的程序却越改越错并且还不能还原到修改前，磁盘空间充足，记得存储你的老版本程序。

最后，如果你的程序出现了很多解释不了的错误，你应该考虑一下是否有更好的方法去编写程序或使用一些更加简洁的算法，这样或许可以正确地实现你的程序。

**编程与翻译感想：**

学习一门计算机语言是不断犯错发现错误改正错误的过程，在学习的过程中，错误也是一种宝贵的经验，我们要学会如何从错误中进步，逐渐掌握一门语言的应用。