**一、作业要求**

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

**二、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
        + 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,
        + 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,

1. One might well wonder why there isn’t a static checker that detected the fact that the line of code *temp.reverse* doesn’t cause any useful computatation to be done, and is therefore likely to be an error.
2. 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.”

* 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.”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!)
  + 1. **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?

37 Arthur Conan Doyle, “The Sign of the Four.”

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.

**三、翻译**

**6.2.3 艰难之路**

肯尼迪总统的父亲约瑟夫˙P˙肯尼迪常常教育他的孩子说：“艰难之路，唯勇者行”。但他从未调试过一个软件。下面这段话则给出了软件调试遇阻时的一些有效提示：

**试看这些常见错误**，比如，你是否

* 弄错了给函数传递参数的顺序？
* 犯了拼写错误，比如，在应该大写时却小写了？
* 没有初始化变量？
* 检查两个浮点数是精确相等“==”而不是约等于“=” ？（请牢记浮点数算数不同于你在学校学的算数）
* 在你想要两对象相等时（例如：id（L1）==id（L2））检查数值是否相等（例如：通过写出表达式L1==L2来比较两列）
* 忘了一些内置函数会产生副作用？
* 忘记“（）”会把对类函数的对象引用变成函数调用？
* 无意中创建了别名？
* 犯了其他你常犯的错误？
* **不要再纠结为什么程序没像你想的那样运行，应该问问自己是什么原因让它会像现在这样运行。**这应该是个比较容易回答的问题，而且也可能是找出调试程序方法的良好开端。
* **请牢记错误一般并不会出现在你所认为地方，**要真在那里的话你早就发现了。一个判断从哪儿开始检查的有效方法就是先找最不可能出错的地方。就像福尔摩斯说的：“排除所有其他可能，剩下的就是真相。”
* **尝试将问题向他人阐述。**所有人都有盲点，很多时候就在你试图去向别人解释问题时你就会发现自己忽略了什么。去解释错误不可能出现在某个地方是一件有益的事情。
* **不要迷信你读到的资料**，尤其是文献。程序可能不会照注释中所说的那样运行。
* **不要一味调试，不妨做点记录**，这样可以帮助你从另一个角度看问题。
* **先放一放明天再试。**虽说这样一来解决问题可能不如对其死缠烂打来的快，但是你用来找错的时间会缩短很多。也就是说，可能是可以用等待来换取效率的。（同学们，这是应该早一些而不是晚一些去做编程错误集的绝佳理由！）

**6.2.3 揪出错的那一刻**

当你发现一个代码错误时，那种想要马上运行一下修正过的程序的冲动是几乎不可抵抗的，但最好还是放慢一点，请记住最终目标不是修改一个错误，而是更快更高效地编出零错误程序。

问问自己这一个错误是否可以解释所有出现的问题，还是仅仅是冰山一角？如果是后者，最好在修改这个错误时协同考虑其他变化。举个例子，假设你发现错误的原因是不小心改变了一个列表，那么你可以局部性地避开这个问题（比如把列表备份），或者可以用元组替代列表（因为元组是不变的），这样或许可以避免程序其他地方出现类似的错误。

在改动之前，应该预想一下修改可能带来的衍生问题：会不会扰乱其他代码？会不会使程序过于复杂？是否有可能整理好代码的其他部分？

请时刻确保你可以找回更改前的代码，没有什么比一系列的更改使得程序与初衷渐行渐远并且无法回到原点更让人沮丧的了。磁盘的空间一般都很充足，可以用它来储存你的旧版程序。

最后，如果有很多无法解释的错误，你或许会怀疑一个个地找错、修改还是不是一种明智做法，那么这时候考虑一下是否有更好的程序组织方法或更容易正确运行的简单算法不失为上策。

**四、心得体会**

第一次接触编程应当是大一初识C++，当时觉得除了高数就当是它最难，但就是因为学习它有困难，所以在成功写出一个程序的时候的喜悦也是难以言表的。大一下开始从基础抓起这门课程的学习也有点起色，也开始产生了兴趣，而兴趣是最大的老师，让自己可以静下心，认真学。还记得最后和老师说谢谢的时候，老师说了一句你最该谢的应当是你自己。编程中无法避免的会遇到bug，有时候只是我们的不小心引起的小错误，解决bug应该是每个程序员的必经过程，每解决一个bug就像是通过了一个游戏关卡离大boss也就越来越靠近了。我觉得编程最大的体会就是**去写**，**去摸索**，**去完善**。努力做到这三步，想学好编程语言应该是不难的。