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.  
翻译

6.2.3在艰难期Joseph P.肯尼迪，肯尼迪总统的父亲，说教他的孩子们，“当事情变得艰难，便艰难的走。”（36他还据说告诉肯尼迪，“别买了一票超过必要的。如果我要投机取巧，我会被诅咒的，但是他从来没有调试过一个软件。此小节包含有关调试时遇到困难是什么做一些务实的提示。当调试变得困难时。  
寻找通常的嫌疑犯。你有  
1将参数传递给错误的顺序中的函数，  
2拼错的名字，例如，输入一个小写字母的时候，你应该输入一个大写的人，  
3未能重新初始化变量，  
4测试，2个浮点值是相等的（= =），而不是几乎相等的（记住，浮点运算是不一样的算术你在学校里学到的），  
5测试值相等性（例如，比较两列表通过写作表达L1 = = L2）当你的意思是对象相等（例如，ID（L1）= = ID（L2）），  
6忘记了一些内置函数有副作用，  
7一个被遗忘的（），这将把一个引用的对象变成一个函数调用，  
8创建一个无意的别名，或  
9犯了一个错误，这是典型的你。  
不要问自己为什么这个计划没有做你想做的事情。相反，问问你自己为什么它是这样做的。这应该是一个更容易回答的问题，并且很可能是解决如何修复程序的一个好的第一步。  
请记住，错误很可能不是你认为它的地方。如果是，你可能会发现它很久以前。一个实际的方式去决定在哪里看都是问哪里的bug是不可能的。就像福尔摩斯所说，“排除所有其他的因素，剩下的就是真相。”（37 arthurconan多伊尔”牌的四”。）  
向别人解释这个问题。我们都在开发盲点。通常情况下，只是试图解释这个问题，仅仅试图解释这个问题的人会导致你看到你已经错过的东西的情况。一个很好的尝试来解释为什么错误不能在某些地方。  
不要相信你读到的每件事。特别是，不要相信文档。该代码可能不会做什么评论建议。  
停止调试并开始编写文档。这将有助于你从不同的角度看问题。  
走开，然后再试一次。这可能意味着错误是固定的时间，如果你已经坚持了它，但你可能会花很多时间寻找它的时间。也就是说，它是可能的贸易延迟的效率。（学生，这是一个很好的理由开始工作的编程问题，而不是后来！）  
6.2.4当你发现的bug  
当你认为你在你的代码中发现了一个错误，开始编码和测试一个固定的诱惑几乎是不可抗拒的。然而，它往往是更好的，放慢了一点。记住，目标不是固定一个错误，而是快速和有效地向一个错误的程序移动。  
问自己，如果这个错误解释了所有的观察到的症状，或者它只是冰山的一角。如果是后者，可以更好地考虑照顾这个错误。举例来说，假设你发现错误是一个意外突变的结果。你能解决这个问题的局部（也许通过复制列表），或者你可以考虑使用一个元组，而不是一个列表（因为元组是不可变的），也许是消除代码中的其他地方类似的错误。  
在作出任何改变，尝试理解分歧所提出的“修复”。它会打破东西吗？是否引入过多的复杂性？它是否提供了机会来整理代码的其他部分？  
总是要确保你能回到你所处的地方。没有什么比认识到一系列的改变让你更进一步的目标，而不是你开始的时候，并没有办法回到你开始的地方。磁盘空间通常很丰富。用它来存储旧版本的程序。  
最后，如果有许多不明原因的错误，你可能会考虑是否找到和修复错误的时间，甚至是正确的做法。也许你会更好地思考是否有更好的方法来组织你的程序或一些简单的算法，将更容易实现正确。

心得体会 :一开始接触这门课觉得很难，像天书一样，根本不明白在说什么。甚至在想我们专业为什么要学习这门课。然后到处理各种作业，总觉得别的同学可以很轻松的完成自己却遇到各种问题，停滞不前，渐渐开始逃避，觉得自己真的是学不好这门课。学期快结束了不得不逼自己去完成作业和考前复习，看完书后，一点一点的尝试慢慢有了信心，觉得自己也可以完成一些简单的编程，虽然运行不了时各种改错很伤，但是运行成功后带来的喜悦不可言喻，作业八里不限制主题的编程觉得很有意思，选择了很多经典算法和小游戏，越来越体会到python语言的简洁和灵活