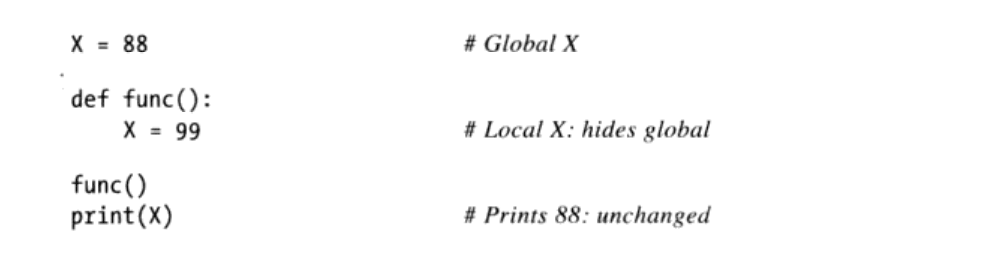
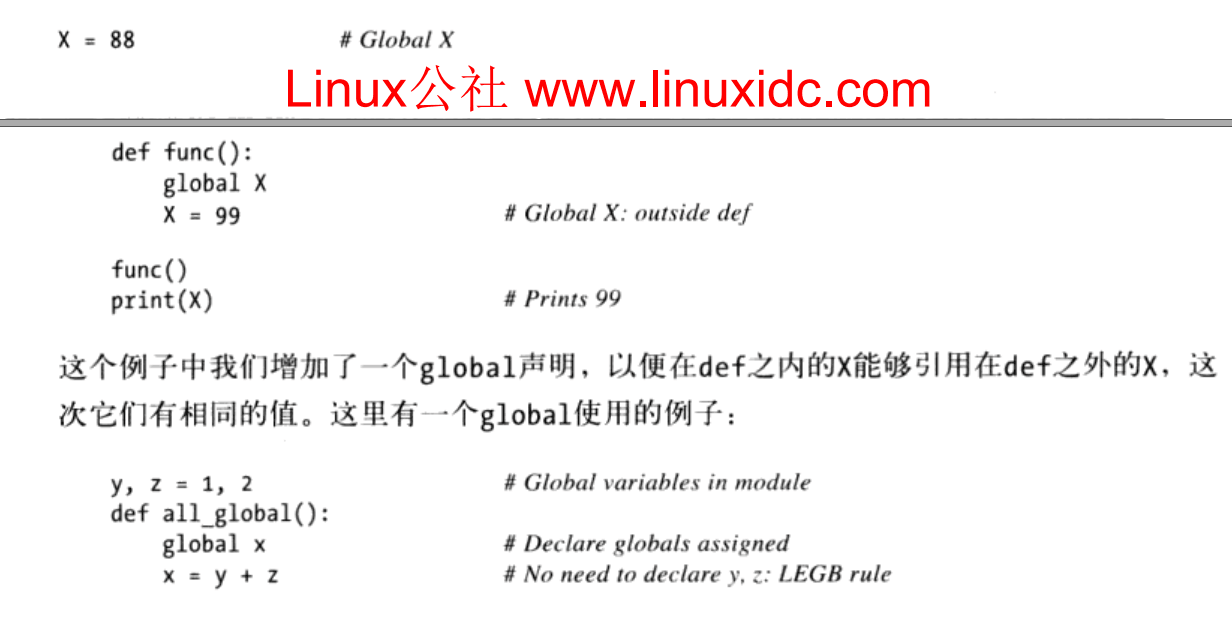
使用global在函数内改变全局变量：



使用global在函数内改变全局变量：

第二个例子中，x在模块中并没有定义，但在函数中声明为全局变量，说明其存活时间只在函数内

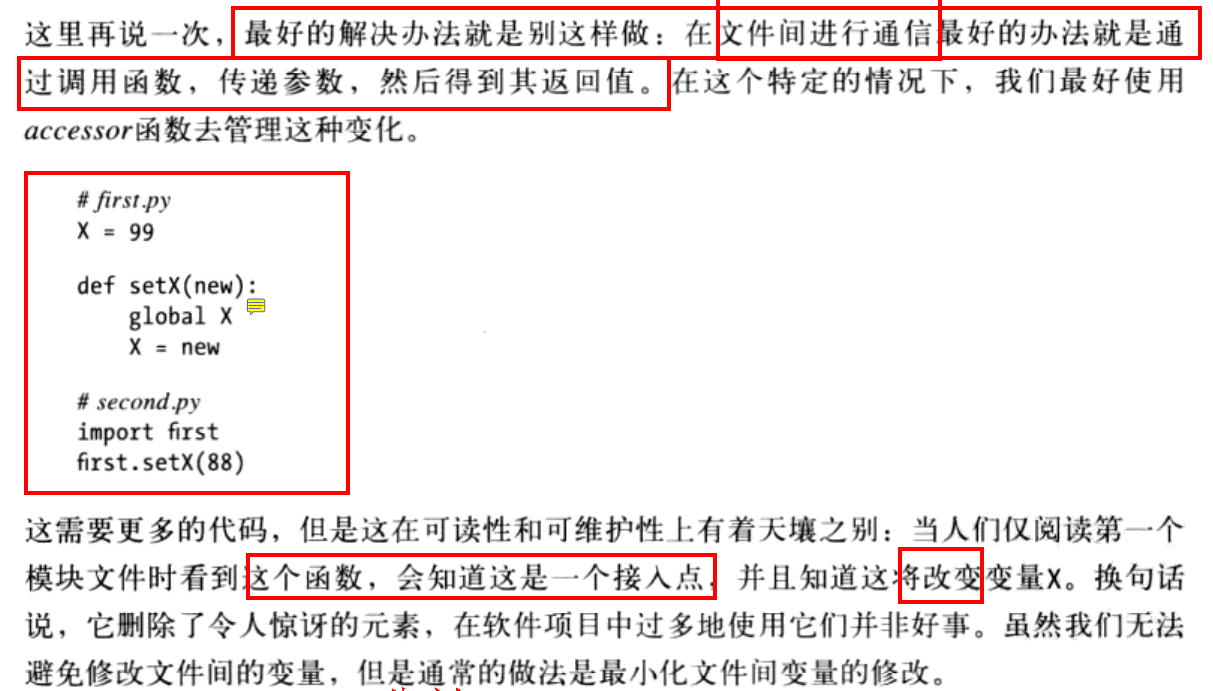


最小化全局变量：如果在多个函数内部修改了全局变量，那么在跟踪全局变量的时候就很麻烦，所以最好不要在多个函数内部使用global声明全局变量。

另外，全局变量在多线程程序中是被当做共享内存来通信的，这个要特别注意。

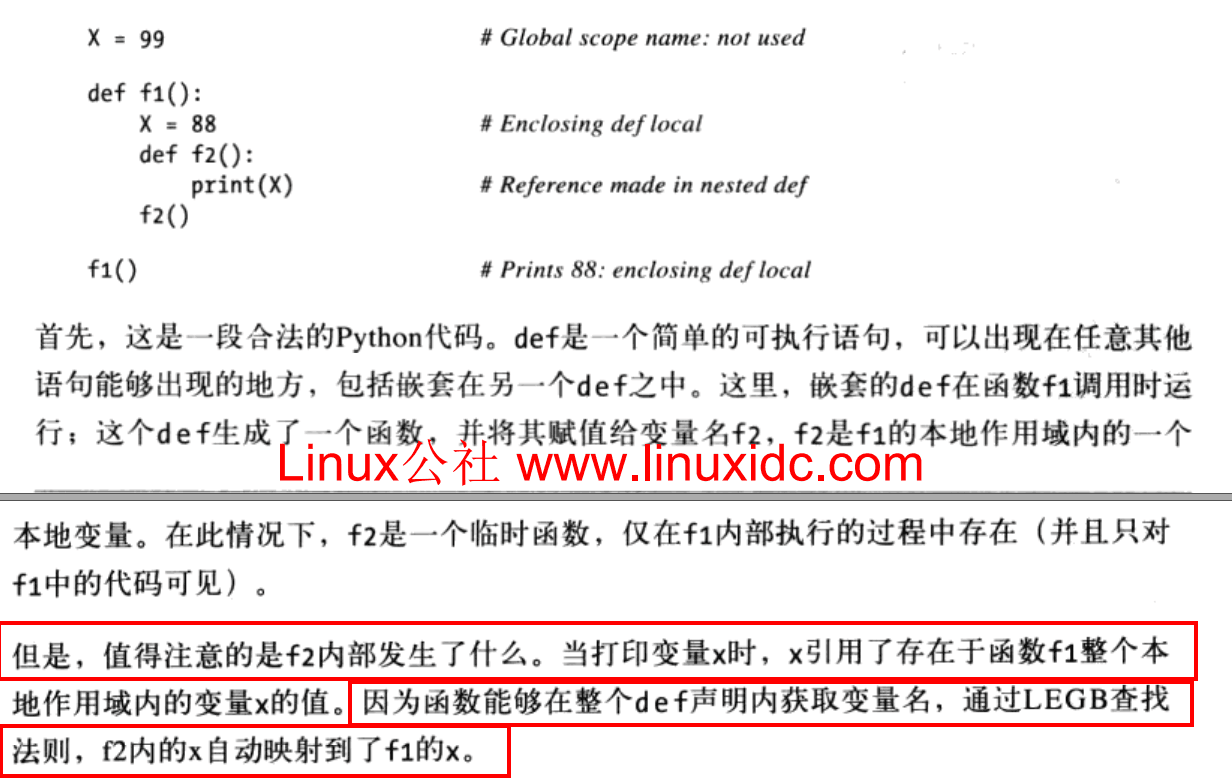


最小化文件间的修改：这里最小化文件间的修改指的是模块的全局变量在import过程中在另外一个模块被修改，这样其他import该模块并使用全局变量的的话就不明确该全局变量的值是什么。

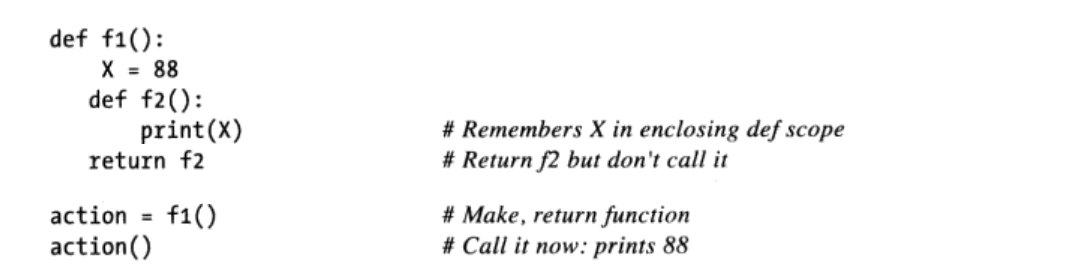
好的方法是使用函数set\_valable\_name（）将全局变量封装在函数中，如果被其他模块导入的话，搜索该函数被使用的位置就知道该全局变量被改变的位置了。

嵌套作用域：

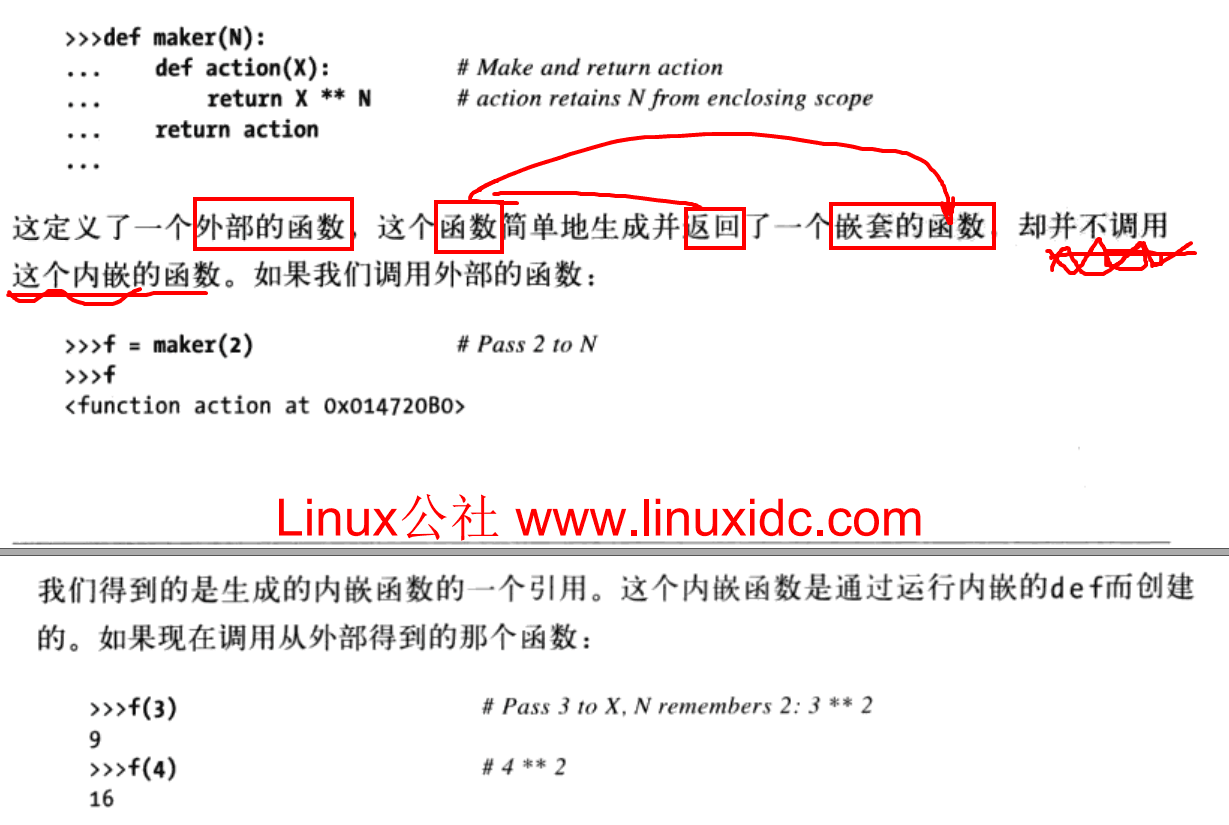
下面的为根据lgdb规则进行搜索，这个没啥好说的



嵌套作用域与返回函数：下面的action = f1()是调用f1函数获取了一个返回值，返回值为返回的函数的地址，然后使用action()才是执行返回的函数，这种使用逻辑要特别注意

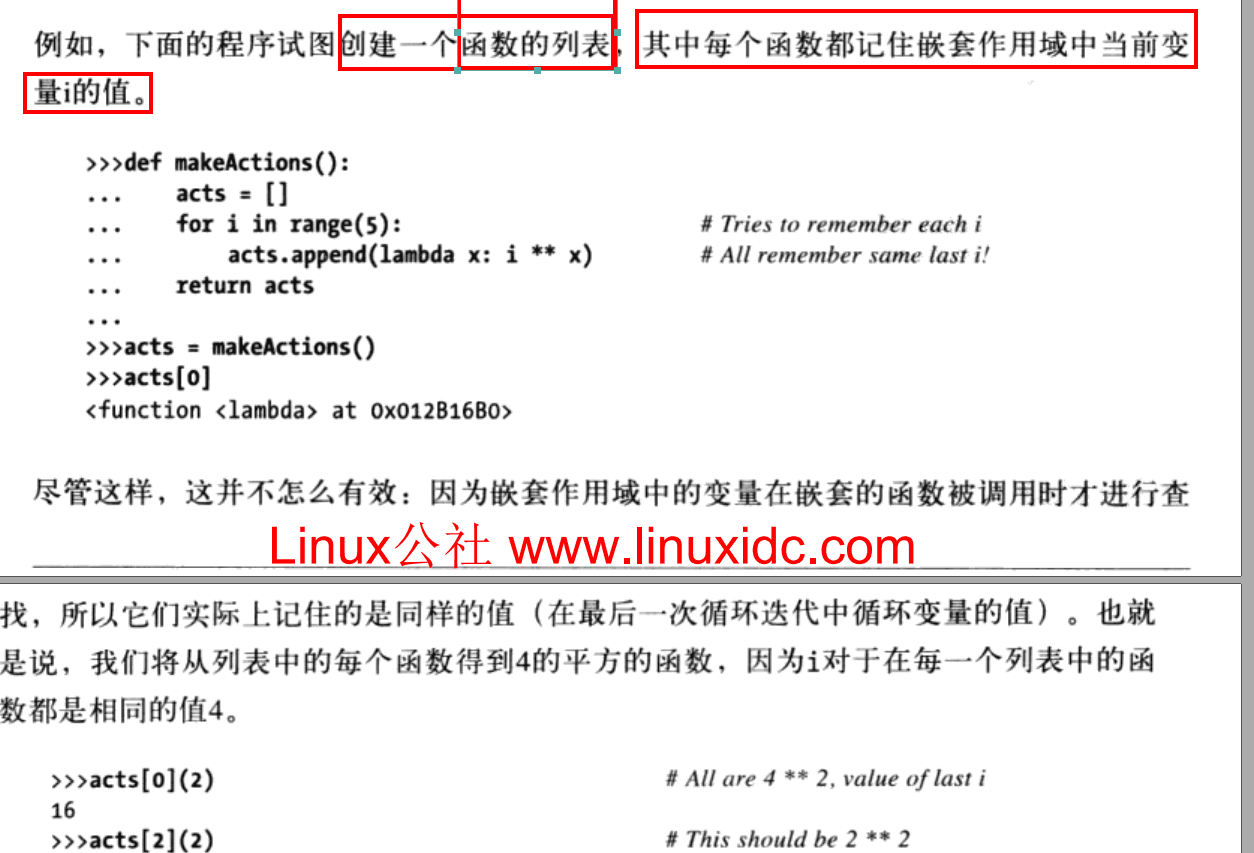


工厂函数，记忆模式。前提是嵌套函数的最外层函数要有一个参数（即最外层函数是带参数函数），然后在内层嵌套的函数使用外层函数的参数。 基本原理是，外层传入的参数会被记住，见下面的例子。最开始f = maker(2)，外层传入的参数被f记住，然后使用f(3)的模式调用maker函数的返回值----action函数。

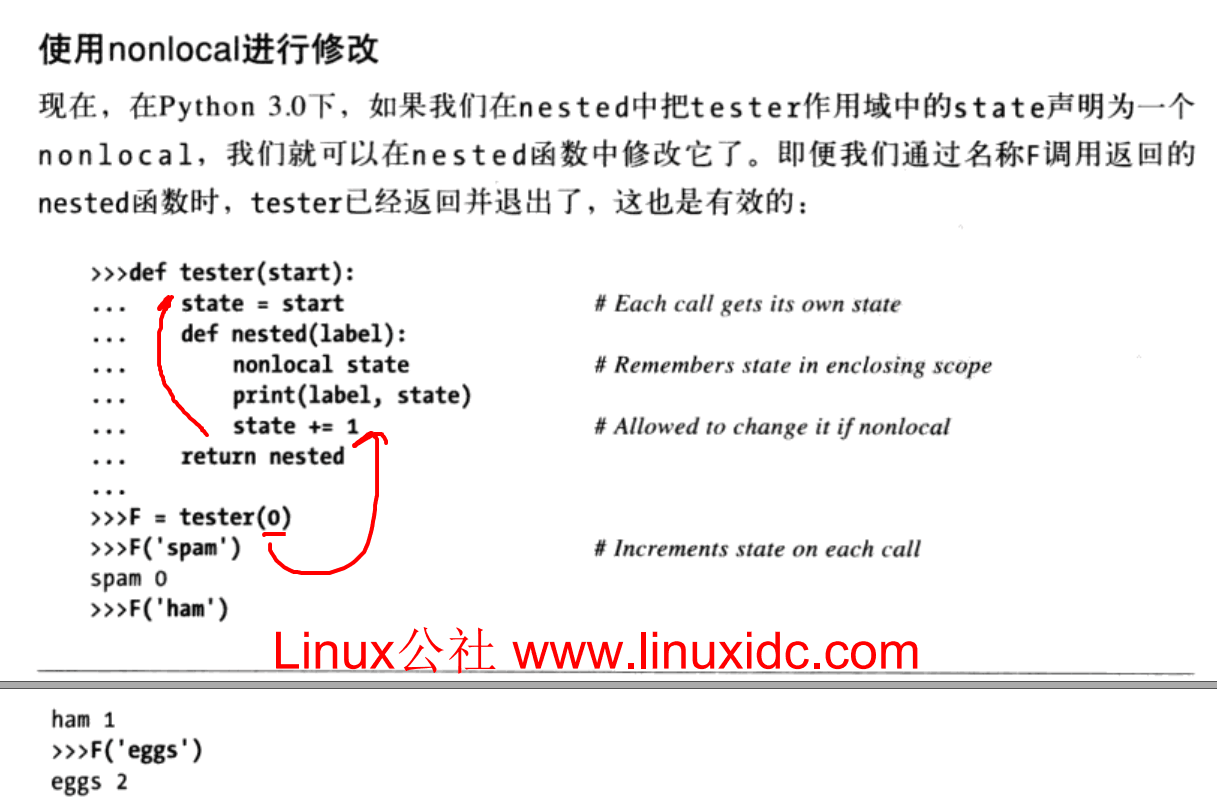


在嵌套函数里面创建函数列表：注意这种函数列表中的函数的使用方式：action[0](2)

这里面还涉及使用lambda创建单语句函数，与函数的默认传参等



使用nolocal对def嵌套函数的变量进行修改。被修改的变量所在的位置：最外层的def函数中的变量，nolocal语句使用的位置：在第二个被嵌套的def函数之中。下面的例子将nolocal和嵌套函数的记忆模式一起结合使用了，要仔细分析



使用函数的属性保持状态