只要是学过C/C++语言的人，肯定会被指针“坑过”，或者是使用了空指针，或者是将指针指向了非法区域并试图访问它。Java宣称，你们解放了，大家都不用学指针啦！于是，小白们欢呼雀跃。殊不知，Java确实没有指针概念，因为它的每个变量都是指针。“NullPointerException”已经太经典，所以无需举例。 还有的人为了躲开讨厌的C语言指针和臃肿的Java代码，学了更易上手的Python。他们可能不知道，Python依然无法避免引入指针概念。下面是一个小例子：  
In [1]: result = ［[0, 0］]\* 3  
In [2]: result  
Out[2]: ［[0, 0], [0, 0], [0, 0］]  
目前看来，代码的输出如你所想。然后，我们尝试为result中的元素赋值：  
In [3]: result[0][0] = 3  
In [4]: result  
Out[4]: ［[3, 0], [3, 0], [3, 0］]  
嗯？输出结果是否跟你想象中的不一样呢？本想只对result中的第0个元素操作，但是result中的所有三个元素都有了变化。再尝试一下：  
In [5]: result[0][1] = 5  
In [6]: result  
Out[6]: ［[3, 5], [3, 5], [3, 5］]  
似乎result中的三个元素的值是同步变化的。其实，不难猜想，result这个list中包含的三个list（初始为[0,0]）其实指向的是一个list实例，而result[0], result[1]以及result[2]都是指向这个实例的指针，所以result中的三个元素是同步变化的。这个猜想可以通过调用id()来查看object的内存地址来验证：  
In [10]: id(result[0])  
Out[10]: 159288460  
In [11]: id(result[1])  
Out[11]: 159288460  
In [12]: id(result[2])  
Out[12]: 159288460  
  
那么如何达到预想的效果呢？即改变result[0]的值，而result[1], result[2]不受影响。可以这样初始化result（笨方法）：  
result = []  
for i in range(3):  
    result.append(list([0,0]))  
这样以来保证了result中的每个元素都是新的list对象，它们的地址当然也是不同的。

再看几个例子:

**实验1：**

1. a=[1,2]
2. b=a
3. a.append(3)
4. print b
5. 输出：[1,2,3]

嗯，这样看起来python的变量都是指针。

**实验2:**

1. a=1
2. b=a
3. a=2
4. print b
5. 输出：1

嗯？why？b不是指针吗？为什么这次b没有跟着a变呢？

来看看下面这个情况吧

**实验3:**

1. a=[1,2]
2. b=a
3. a=[3]
4. print b
5. 输出：[1,2]

阿哈！这次b又没有跟着a变，为什么呢？   
其实我们的猜想是正确的，python的变量的确都是指针，只是在重新被附值时，指向了另一个list，而不是像情况1那样，改变了那个list。   
在情况2也是这样，第三句并不是a指向的1改变了，而是a的指向改变了。原来的1还在那里，没变。

**那么，如果真的想复制list怎么办？**

1. b = a [:]