

Re: Python List Issue

Source: <http://coding.derkeiler.com/Archive/Python/comp.lang.python/2005-04/msg00096.html>

- *From:* Greg Ewing <greg@xxxxxxxxxxxxxxxxxxxxxxxx>
 - *Date:* Thu, 31 Mar 2005 17:34:14 +1200
-

Nick L wrote:

```
I noticed that with python lists, generally when you
make a copy of a list (ie, List1 = List2) List1 just becomes a reference to
List2 and any modifications done to List1 affects List2. Ok I can live with
this but I want to make a completely seperate copy not attached to the
original in anyway.
```

Although your immediate question has been answered (you wanted a deep copy, not a shallow one), I think you should take a step back and consider whether you really need to do that much copying in the first place.

With the proper approach, I find it's extremely rare to need to copy anything in a Python program, and rarer still to need deep copying. Generally, it's better for a function not to modify objects passed to it, unless that's the purpose of the function. If a function needs a local copy of an argument, it should create one itself, and not put the burden on the caller.

Also, there are some things about your code that suggest you still do not fully understand how scoping and assignment work in Python:

```
def ApplySourceFunc(bob, final):

> ...

    bob = final
    final = []
```

These last two statements do not accomplish anything useful. All they do is change which objects are referred to by the parameter names 'bob' and 'final', which are local to the function, and have no effect on anything outside it.

Re: Python List Issue

```
def ApplyOperatorsLoop(aList):  
  
> ...  
  
    final.append(iTemp)  
    ...
```

Here you are referring to the global variable 'final', not the one passed as a parameter to ApplySourceFunc(). To get the effect you seem to be after, you would need to pass 'final' on as a parameter to ApplyOperatorsLoop().

Here is how I would write this:

```
def ApplySourceFunc(bob, final):  
    for item in bob:  
        ApplyOperatorsLoop(item, final)  
  
def ApplyOperatorsLoop(item, final):  
    new_item = AddGetSpeed(item)  
    if new_item:  
        final.append(temp)  
  
def AddGetSpeed(tList):  
    if tList and tList[-1][0] == 0:  
        new_tList = tList[:]  
        new_tList[-1] = new_tList[-1][:]  
        new_tList[-1][0] = "GetSpeed()"  
        print "New Rule 'GetSpeed()' Added to the List"  
        return new_tList  
    else:  
        return None
```

Notice how all the copying is done inside AddGetSpeed(), it only copies what is needed, and the fact that the copying is needed is encapsulated within the function; its callers don't need to know.

I have also made use of some other common Python idioms:

Re: Python List Issue

- * Whenever possible, it is simpler and clearer to iterate directly over the items of a list, rather than iterating over its indices and then indexing it.

- * Almost any object can be used as a truth value. Non-empty lists count as true; empty lists and None count as false.

- * The 'and' and 'or' operators short-circuit: if the first operand determines the result, the second operand is not evaluated.

- * Negative list indices are counted from the end of the list; e.g. aList[-1] means the last item of aList.

Hope that helps,

--

Greg Ewing, Computer Science Dept,
University of Canterbury,
Christchurch, New Zealand
<http://www.cosc.canterbury.ac.nz/~greg>

.