```
# -*- coding: utf-8 -*-
"""
Created on Tue Oct  6 10:20:38 2015
@author: hina
Reference:
www.jeffknupp.com/blog/2012/11/13/is-python-callbyvalue-or-callbyreference-neither
"""

# In Python, (almost) everything is an object.

# There are two kinds of Objects in Python - Mutable and Immutable.
# A Mutable Object exhibits time-varying behavior.
#     Changes to a Mutable Object are visible through all Names Bound to it.
#     Python's lists are an example of mutable objects.
# An Immutable Object does not exhibit time-varying behavior.
#     The value of Immutable Objects can not be modified after they are created.
#     They can however be used to compute the values of new objects.
#     Strings and Integers are examples of immutalbe objects.

# list objects are mutable
# this is allowed
myList = ['Jill', 'Jane', 'Harry']
myList[0] = 'Tom'

# string objects are immutable
# this is not allowed
#myStr = 'Rick'
#myStr[0] = 'N'

# What we commonly refer to as "variables" in Python are more properly called Names.
# Likewise, "assignment" is really the Binding of a Name to an Object.
# Each Binding has a Scope that defines its visibility (usually the Block in which the
# Name originates.

# Let's look at an example of Names and Object Binding

# (1) The Name some_guy is Bound to the String Object 'Fred'
some_guy = 'Fred'

# (2) The Name fisrt_names is Bound to the List Object []
first_names = []

# (3) The 0th Element of the List Object created above now contains
#     the String Object 'Fred'
first_names.append(some_guy)

# (4) The Name another_list_of_names is also now Bound to the List Object created above
another_list_of_names = first_names

# (5) The 1st Element of the List Object created above now contains
#     the String Object 'George'
another_list_of_names.append('George')

# (6) The Name some_guy is now Bound to the String Object 'Bill'
#     (and not Bound to String Object 'Fred' anymore)
some_guy = 'Bill'
```

```
# So this will now yield 'Bill', ['Fred, 'George'], ['Fred, 'George']
print (some_guy, first_names, another_list_of_names)

# Note: in all, there were 3 Names, 1 List Object, and 3 String Objects in eg above

print ()

# another thing of note is copying versus binding:

# this will simply bind a and b to the same object
# you can test this with b is a (which will evaluate to True below)
# this also means that changes to b will be reflected in a
a = [1, 2, 3]
b = a
print (b is a)
b[2] = 10
print (a, b)

# this will create a copy of a, so a and c now point to different objects
# you can test this with c is a (which will evaluate to False below)
# this also means that changes to c will not be reflected in a
a = [1, 2, 3]
c = []
c[:] = a [:]
print (c is a)
c[2] = 10
print (a, c)

print ()
```