Neuer Verweis ist nicht gleich neue Kopie.

```
# Achtung, "Falle":
# [0]*3 erzeugt eine Liste der Länge drei, deren Einträge
# konstant gleich Null sind.
# [[0]*3]*3 erzeugt eine Liste der Länge drei, deren Einträge
# dreimal DIESELBE Liste [0]*3 sind - das sieht zwar aus wie
# eine 3 mal 3 Matrix ...
tabelle = [[0]*3]*3
print('1:')
for zeile in tabelle:
    print(zeile)
# ... aber die Zuordnung eines neuen Eintrags erfolgt immer
# gleichzeitig in ALLEN Zeilen (die ja nur Verweise auf
# DIESELBE Zeile sind!), also:
tabelle[1][1] = 42
print('2:')
for zeile in tabelle:
    print(zeile)
# Eine Matrix mit "wirklich verschiedenen Einträgen" erhält man
# z.B. so:
tabelle = [list(range(3*i,3*i+3)) for i in range(3)]
print('3:')
for zeile in tabelle:
    print(zeile)
tabelle[1][1] = 42
print('4:')
for zeile in tabelle:
    print(zeile)
# Oder man verwendet die Funktion deepcopy, die eine ECHTE, NEUE
# KOPIE des Objekts erzeugt (nicht nur einen neuen Verweis auf
# dasselbe Objekt):
zeilen_objekt = [0]*3
tabelle = [deepcopy(zeilen_objekt) for i in range(3)]
tabelle[0][0] = 42
print('5:')
for zeile in tabelle:
    print(zeile)
# Das in allen Details darzustellen würde hier den Rahmen
# sprengen: Wenn etwas nicht wie erwartet funktioniert, sollte man aber an den
# wichtigen Unterschied zwischen "neuen Verweisen auf dasselbe Objekt" und
# "neue Kopie eines Objekts" denken. (Matrizen sind aber vielfach besser
# mit der Bibliothek numpy zu realisieren, dazu kommen wir später noch.)
```