mutability and misc.

Ben Bolker

09:06 21 January 2015

Mutability

We hit a snag on Monday with our matrix construction due to something called **mutability**. That is, when Python assigns an object to another object, sometimes it only copies a *reference* to the other object. In particular, lists are mutable (all the other kinds of things we've seen — integers, floats, characters — are immutable, and thus less confusing).

For example:

```
x = [0,0,0]
y = x
y[0] = 2 ## change y
print(x,y)
```

```
## [2, 0, 0] [2, 0, 0]
```

If we really want to copy an object we can make a **deep copy**:

```
x = [0,0,0]
import copy
y = copy.deepcopy(x)
y[0] = 1
print(x,y)
```

```
## [0, 0, 0] [1, 0, 0]
```

The `id()` function can be useful to check whether two objects are really the same.

```
x = [0,0,0]
import copy
y1 = copy.deepcopy(x)
y2 = x
print(y1 == x,
      y2 == x,
      id(y1) == id(x),
      id(y2) == id(x))
```

```
## True True False True
```

How does this apply to our matrix construction problem?

```
def matlist0(m,n):
    return([[0]*n]*m)

def matlist1(m,n):
   x = []
   for i in range(m):
      x.append([])
      for j in range(n):
          x[i] += [0]
   return(x)

def matlist2(m,n):
   x = [[]]*m ## multiply _empty_ list
   for i in range(m):
      x[i] = [0]*n
   return(x)
```

The first one fails to make copies, the second and third both succeed. (You can use `id()`, or try setting a value, to check.)

Also handy, from StackOverflow:

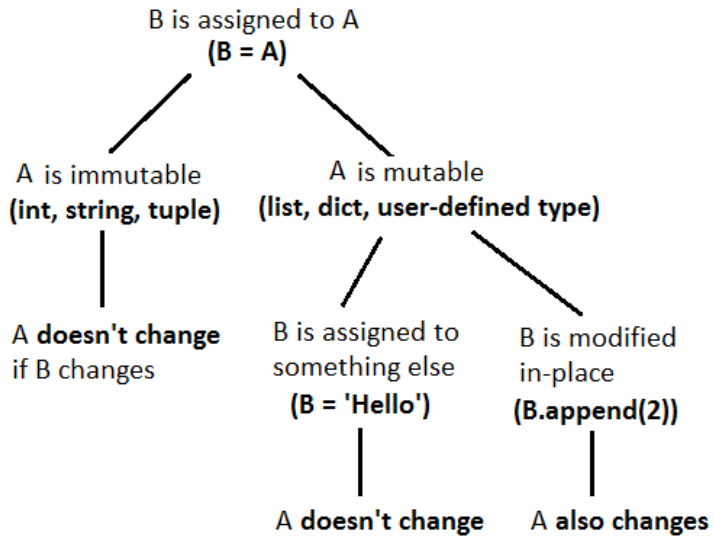

Figure 1: mutability mnemonic

- Another StackOverflow question