12. Shallow và deep copy

Như chúng ta đã thấy trong chương "Các kiểu dữ liệu và biến", Python có một hành vi lạ - so với các ngôn ngữ lập trình khác - khi gán và sao chép các kiểu dữ liệu đơn giản như số nguyên và chuỗi. Sự khác biệt giữa shallow và deep copy chỉ có liên quan đối với các đối tượng kết hợp, đó là các đối tượng chứa các đối tượng khác, chẳng hạn như đối tượng kiểu list hoặc các thể hiện lớp.

Các bạn có thể quan sát qua ví dụ sau. Trong đoạn mã biến y chỉ đến cùng vị trí bộ nhớ của biến x. Điều này thay đổi, khi chúng ta gán một giá trị khác cho y. Trong trường hợp này, y sẽ nhận được vị trí bộ nhớ riêng, như chúng ta đã thấy trong chương "Các kiểu dữ liệu và biến".

```
>>> x = 10

>>> y = x

>>> id(x)

38758384L

>>> id(y)

38758384L

>>> y = 9

>>> id(y)

38758408L

>>> id(x)

38758384L

>>>
```

Nhưng ngay cả khi hành vi nội bộ này có vẻ lạ so với các ngôn ngữ lập trinh khác như C, C++ và Perl, với ví dụ phía trên nó đáp ứng được mong muốn của chúng ta. Nhưng nó có thể là vấn đề, nếu chúng ta sao chép các đối tượng có thể thay đổi như danh sách (list) hay dictionary.

Python tạo ra các bản sao thực chỉ khi nó buộc phải làm, tức là nếu lập trình viên yêu cầu nó một cách rõ ràng.

Tôi sẽ giới thiệu cho bạn một số vấn đề, có thể xảy ra khi sao chép các đối tượng có thể thay đổi, tức là khi sao chép list hay dictionary.

Sao chép một list.

Ví dụ: Kết quả của các phép gán items1 và items 2 đều đúng như mong muốn của chúng ta. Đầu tiên items1 trỏ đến vùng nhớ có danh sách ["sach","but"], bước items2 = items1 chính là bước để items2 cũng trỏ đến vùng nhớ chứa ["sach","but"], vậy không có vùng nhớ nào được tạo ra thêm ở thời điểm này. Sau đó, items2 được trỏ đến một danh sách khác ["bong den","ban ghe"], đây là một vùng nhớ khác. Good!!!

```
items1 = ["sach","but"]
items2 = items1
items2 = ["bong den","ban ghe"]
```

```
print "items1 = ", items1
print "items2 = ",items2

Output:

items1 = ['sach', 'but']

items2 = ['bong den', 'ban ghe']
```

Ví dụ: Tiếp theo, sau bước gán items2= items1 thì cả hai đều trỏ vào cùng một vùng nhớ chứa danh sách ["sach","but"]. Câu hỏi là nếu ta thay đổi giá trị thuộc items1 hoặc items2 thì điều gì sẽ xảy ra? Các bạn có thể thấy ngay khi thay đổi thông qua items2 hoặc items1 thì danh sách đó cũng thay đổi.

```
items1 = ["sach","but"]
items2 = items1  # shallow copy
items2[0] = "ban"
print "items1 = ", items1
print "items2 = ",items2

Output:

items1 = ['ban', 'but']

items2 = ['ban', 'but']
```

Cả hai ví dụ trên cho ta cái nhìn về shallow copy ở phép gán như items2 = items1.

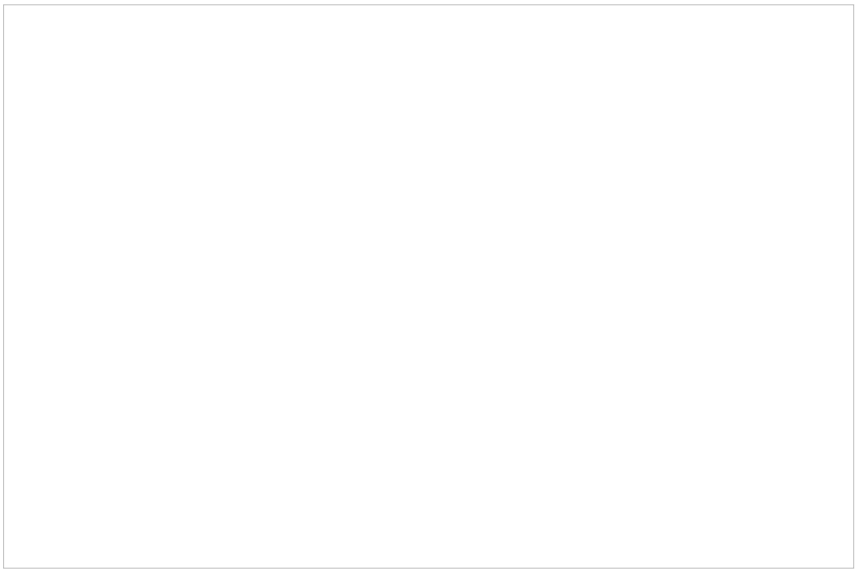
Slicing là một giải pháp cho shallow copy ở bên trên, nó sẽ không gây ra “side effect” cho ví dụ trước đó.

```
items1 = ["sach","but"]
items2 = items1[:]
items2[0] = "ban"
print "items1 = ", items1
print "items2 = ", items2

Output

items1 = ['sach', 'but']

items2 = ['ban', 'but']
```

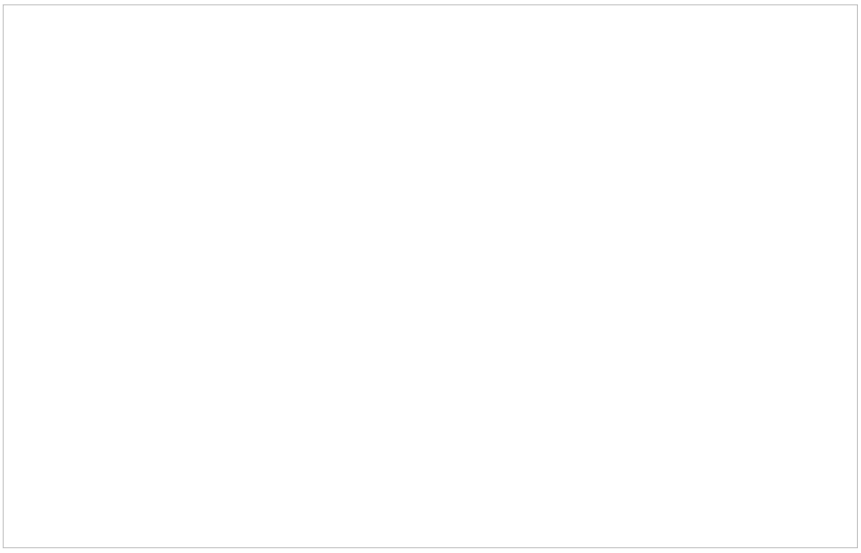
Tuy nhiên, khi dùng slicing thì vẫn sẽ gặp vấn đề nếu list lại chứa sublist như ví dụ dưới đây: lst1 sẽ bị thay đổi nếu lst2 bị thay đổi. Hai hình vẽ dưới mô tả quá trình đó.

```
lst1 = ['a','b',['ab','ba']]
lst2 = lst1[:]
print "lst1 = ", lst1
print "lst2 = ", lst2

lst2[0] = 'c'
lst2[2][1] = 'd'

print "lst1 = ", lst1
print "lst2 = ", lst2

Output:

lst1 = ['a', 'b', ['ab', 'ba']]

lst1 = ['a', 'b', ['ab', 'ba']]
```

```
lst1 = ['a', 'b', ['ab', 'd']]

lst1 = ['c', 'b', ['ab', 'd']]
```



=>



Ta sử dụng phương thức deepcopy để giải quyết vấn đề trên triệt để.

```
from copy import deepcopy
lst1 = ['a','b',['ab','ba']]
lst2 = deepcopy(lst1)
lst2[2][1] = "d"
lst2[0] = "c"
print "lst1 = ", lst1
print "lst2 = ", lst2


Output:


lst1 = ['a', 'b', ['ab', 'ba']]

lst2 = ['c', 'b', ['ab', 'd']]
```

Kết Luận

Về bản chất “Shallow” chính là một phép gán thông thường trong Python. Chúng ta dùng hai cái tên khác nhau để trỏ đến cùng một vùng nhớ. Nếu đối tượng chúng ta muốn sao chép là những đối tượng có thể bị thay đổi như list, dictionary thì các bạn nên sử dụng deep copy để tránh những “side effect” mà dùng “shallow” sẽ gặp phải khi các đối tượng bị thay đổi trong chương trình.

Như vậy, cho đến bài học này, chúng ta đã tìm hiểu qua hết các vấn đề cơ bản về biến cũng như các kiểu dữ liệu trên biến (bao gồm cách khai báo, toán tử và các phương thức hỗ trợ đối với mỗi kiểu dữ liệu). Trong bài học tiếp theo, chúng tôi sẽ trình bày một nội dung mới, đó là về hàm trong Python (Function).