How to insert a dictionary in another dictionary in Python (How to merge two dictionaries)

There are two ways to "insert a dictionary in another dictionary".

Merge Dictionaries

One of them is to merge the two. All the keys of of one of the dictionaries become also the keys of the other dictionary. Because how this works, it actually does not matter if the dictionary where the result will be is the same as one of the original ones, or if it is a third dictionary.

examples/python/merge_dictionaries.py

```
1.  from __future__ import print_function
2.
3.  team_a = {
4.      'Foo' : 3,
5.      'Bar' : 7,
6.      'Baz' : 9,
7.  }
8.
9.
10. team_b = {
11.     'Moo' : 10,
12.     'Boo' : 20,
13.     'Foo' : 30,
14. }
15.
16. print(team_a)       # {'Baz': 9, 'Foo': 3, 'Bar': 7}
17. print(team_b)       # {'Foo': 30, 'Moo': 10, 'Boo': 20}
18.
19. team = dict(team_a.items() + team_b.items())
20. print(team)         # {'Bar': 7, 'Foo': 30, 'Baz': 9, 'Boo': 20,
    'Moo': 10}
21.
22.
23. team["Foo"] = 100
```

```
print(team)        # {'Bar': 7, 'Foo': 100, 'Baz': 9, 'Boo': 20, 'Moo': 10}
print(team_a)      # {'Baz': 9, 'Foo': 3, 'Bar': 7}
print(team_b)      # {'Foo': 30, 'Moo': 10, 'Boo': 20}
```

As you can see there was a key that appeared in both dictionaries. For that particular key, in the resulting dictionary we got the value of the appropriate value form last (or right most) dictionary. (In our case that is team_b.)

We used the items method of the >dictionary object that returns a list of tuples. Each tuple holding one key-value pair from the dictionary. Then we take the two lists of tuples, add them together using the +operator. If we added the following code in the middle of our original script we could see that after the addition, we still have 6 tuples. We still have two tuples where the first element is 'Foo'.

```
print(team_a.items())       # [('Baz', 9), ('Foo', 3), ('Bar', 7)]
print(team_b.items())       # [('Foo', 30), ('Moo', 10), ('Boo', 20)]
print(team_a.items() + team_b.items())
    # [('Baz', 9), ('Foo', 3), ('Bar', 7), ('Foo', 30), ('Moo', 10), ('Boo', 20)]
```

Then we turn this list of tuples into a dictionary using the dict function. At this point the second value of the 'Foo' key overwrites the first one in the new dictionary.

So far this is ok. In order to see if these dictionaries are connected or not we can assign a new value to the 'Foo' key of the common dictionary: team["Foo"] = 100 and then we check the content of the 3 dictionaries. The output shows that only the team dictionary has changed. The other two remained with the original values. This means the merging of the two dictionaries actually created a totally separate third dictionary.

Insert dictionary into another dictionary

The other way of insertion requires a new key in one of the dictionaries and the value will be the other dictionary.

examples/python/insert_dictionary.py

```
from __future__ import print_function

team_a = {
    'Foo' : 3,
```

```
5.      'Bar' : 7,
6.      'Baz' : 9,
7.  }
8.
9.
10. team_b = {
11.     'Moo' : 10,
12.     'Boo' : 20,
13.     'Foo' : 30,
14. }
15.
16. print(team_a)    # {'Baz': 9, 'Foo': 3, 'Bar': 7}
17. print(team_b)    # {'Foo': 30, 'Moo': 10, 'Boo': 20}
18.
19. team_a["b"] = team_b
20. print(team_a)    # {'Baz': 9, 'b': {'Foo': 30, 'Moo': 10,
   'Boo': 20}, 'Foo': 3, 'Bar': 7}
21.
22.
23.
24. team_b["Foo"] = 200
25. print(team_b)    # {'Foo': 200, 'Moo': 10, 'Boo': 20}
26. print(team_a)    # {'Baz': 9, 'b': {'Foo': 200, 'Moo': 10,
   'Boo': 20}, 'Foo': 3, 'Bar': 7}
27.
```

In this case we assigned the team_b dictionary to a new key in the team_a dictionary.team_a["b"] = team_b

In the result we can see that team_a has now 4 keys. The 3 it had earlier and the new key b, but the keys from team_b have not been **merged**. It became an internal dictionary. team_a became a (partially) 2-dimensional dictionary. If you wish. The key 'Foo' exists both in the external dictionary, and in the internal dictionary and they hold different values. They are not related at all.

Once that was done we used the same experiment as earlier and changed the content of 'Foo' key of the team_b dictionary using team_b["Foo"] = 200.

The resulting printout shows that both team_b, and the internal part of team have changed. That's because in this case we assign a reference to the dictionary. So when we assigned team_b to team_a["b"] we have not copied the content of team_b, we just connected the existing dictionary to another place where it can be accessed from.