Making Copies of Dictionaries

There are actually 3 kinds of copies you can make and there is an important distinction best illustrated by an example:

- **Use the = sign**

Both dictionaries share the same memory and a change to one means the same change is made to the other.

```python
d1 = {'a': 1, 'b': 2, 'c': 3}
d2 = d1
   |----> d1 is {'a': 1, 'b': 2, 'c': 3}
   |----> d2 is {'a': 1, 'b': 2, 'c': 3}
d1['a'] = 5
   |----> d1 is {'a': 5, 'b': 2, 'c': 3}
   |----> d2 is {'a': 5, 'b': 2, 'c': 3}
```

So when modifying one of the dictionaries, both are modified because `d2` is basically just a pointer to the same thing `d1` is pointing to.

- **Using copy() to make a copy of dictionaries with a simple structure**

This will make a copy of first-level objects (things that do not have their own structure, ie strings, ints, floats). Now, next example:

```python
d1 = {'a': 1, 'b': 2, 'c': 3}
d2 = d1.copy()
   |----> d1 is {'a': 1, 'b': 2, 'c': 3}
   |----> d2 is {'a': 1, 'b': 2, 'c': 3}
d1['a'] = 5
   |----> d1 is {'a': 5, 'b': 2, 'c': 3}
   |----> d2 is {'a': 1, 'b': 2, 'c': 3}
```

Notice that we are making a shallow copy using `.copy()`, so we might expect that `d2` should point to the same thing `d1` is pointing to. So modifying one would mean modifying the other. However, this is not quite right. A shallow copy makes a copy of the first-level data members and only points to the same structure of any second level-data members -- second level data members would include things like lists and dictionaries.
Using copy() to make a copy of dictionaries with a more complex structure

Shallow copy does not make a copy of second-level objects (things that do have structure, ie lists, dictionaries). So let's rewrite the previous example to use second-level data members. Let's put in a list for one of our dictionary values:

d1 = {'a': [7,8,9], 'b': 2, 'c': 3}
d2 = d1.copy()

|-----> d1 is {'a': [7, 8, 9], 'b': 2, 'c': 3}
|-----> d2 is {'a': [7, 8, 9], 'b': 2, 'c': 3}

d1['a'][0]=6

|-----> d1 is {'a': [6, 8, 9], 'b': 2, 'c': 3}
|-----> d2 is {'a': [6, 8, 9], 'b': 2, 'c': 3}

d2['a'].append(10)

|-----> d1 is {'a': [6, 8, 9, 10], 'b': 2, 'c': 3}
|-----> d2 is {'a': [7, 8, 9, 10], 'b': 2, 'c': 3}

Notice that any change we made to d1 or d2's second-level objects (the list) was echoed in the other because the shallow copy did not actually make a full copy of the list and its elements for d2. It only made a pointer to the same list that d1 had in that place.

Using copy.deepcopy(), make a copy of everything, of all the structure

So finally, a deep copy copies everything. There is nothing shared between the two lists anymore. So the previous example will be:

import copy
d1 = {'a': [7,8,9], 'b': 2, 'c': 3}
d2 = copy.deepcopy(d1)

|-----> d1 is {'a': [7, 8, 9], 'b': 2, 'c': 3}
|-----> d2 is {'a': [7, 8, 9], 'b': 2, 'c': 3}

d1['a'][0]=6

|-----> d1 is {'a': [6, 8, 9], 'b': 2, 'c': 3}
|-----> d2 is {'a': [7, 8, 9], 'b': 2, 'c': 3}

d2['a'].append(10)

|-----> d1 is {'a': [6, 8, 9, 10], 'b': 2, 'c': 3}
|-----> d2 is {'a': [7, 8, 9, 10], 'b': 2, 'c': 3}

Notice that d1 and d2 act as completely separate entities, and changes to one do not get made in the other.
This concept also applies to lists of lists, as illustrated by the transcript below:

```python
>>> list1 = [1, 2]
>>> list2 = [3, 4]
>>> list3 = [list1, list2]
>>> import copy
>>> shallow1 = copy.copy(list3) # one way to get a shallow copy
>>> list3
[[1, 2], [3, 4]]
>>> shallow1
[[1, 2], [3, 4]]
>>> shallow1[0][0] = 5
>>> list3
[[5, 2], [3, 4]]
>>> shallow1
[[5, 2], [3, 4]]
>>> shallow2 = list3[:] # another way to get a shallow copy
>>> shallow2[1][0] = 6
>>> list3
[[5, 2], [6, 4]]
>>> shallow2
[[5, 2], [6, 4]]
>>> deep = copy.deepcopy(list3) # how you get a deep copy
>>> deep[0][1] = 7
>>> deep
[[5, 7], [6, 4]]
>>> list3
[[5, 2], [6, 4]]
```