Copy in Python (Deep Copy and Shallow Copy)

In Python, Assignment statements do not copy objects, they create bindings between a target and an object. When we use the = operator, It only creates a new variable that shares the reference of the original object. In order to create “real copies” or “clones” of these objects, we can use the copy module in Python.

Syntax of Python Deepcopy

> **Syntax:** copy.deepcopy(x)

Syntax of Python Shallowcopy

> **Syntax:** copy.copy(x)

Example:

In order to make these copies, we use the copy module. The copy() returns a shallow copy of the list, and deepcopy() returns a deep copy of the list. As you can see that both have the same value but have different IDs.

Example: This code showcases the usage of the `copy` module to create both shallow and deep copies of a nested list `li1`. A shallow copy, `li2`, is created using

copy.copy(), preserving the top-level structure but sharing references to the inner lists. A deep copy, `li3`, is created using **copy.deepcopy()**, resulting in a completely independent copy of `li1`, including all nested elements. The code prints the IDs and values of `li2` and `li3`, highlighting the distinction between shallow and deep copies in terms of reference and independence.

```
import copy
li1 = [1, 2, [3, 5], 4]
li2 = copy.copy(li1)
print("li2 ID: ", id(li2), "Value: ", li2)
li3 = copy.deepcopy(li1)
print("li3 ID: ", id(li3), "Value: ", li3)
```

Output:

```
li2 ID:  2521878674624 Value:  [1, 2, [3, 5], 4]
li3 ID:  2521878676160 Value:  [1, 2, [3, 5], 4]
```

What is Deep copy in Python?

A deep copy creates a new compound object before inserting copies of the items found in the original into it in a recursive manner. It means first constructing a new collection object and then recursively populating it with copies of the child objects found in the original. In the case of deep copy, a copy of the object is copied into another object. It means that **any changes** made to a copy of the object **do not reflect in the original object.**

Deep copy in Python
Image not found or type unknown

Example:

In the above example, the change made in the list **did not affect** other lists, indicating the list is deeply copied.

This code illustrates deep copying of a list with nested elements using the `copy` module. It initially prints the original elements of `li1`, then deep copies them to create `li2`. A modification to an element in `li2` does not affect `li1`, as demonstrated by the separate printouts. This highlights how deep copying creates an independent copy, preserving the original list's contents even after changes to the copy.

```
import copy
li1 = [1, 2, [3,5], 4]
li2 = copy.deepcopy(li1)
print ("The original elements before deep copying")
for i in range(0,len(li1)):
    print (li1[i],end=" ")

print("\r")
li2[2][0] = 7
print ("The new list of elements after deep copying ")
for i in range(0,len( li1)):
    print (li2[i],end=" ")

print("\r")
print ("The original elements after deep copying")
for i in range(0,len( li1)):
    print (li1[i],end=" ")
```

Output:

```
The original elements before deep copying
1 2 [3, 5] 4
The new list of elements after deep copying
1 2 [7, 5] 4
The original elements after deep copying
1 2 [3, 5] 4
```

What is Shallow copy in Python?

A shallow copy creates a new compound object and then references the objects contained in the original within it, which means it constructs a new collection object and then populates it with references to the child objects found in the original. The copying process does not recurse and therefore won't create copies of the child objects themselves. In the case of shallow copy, a reference of an object is copied into another object. It means that **any changes** made to a copy of an object **do reflect** in the original object. In python, this is implemented using the "**copy()**" function.

Shallow copy in Python
Image not found or type unknown

Example:

In this example, the change made in the list **did affect** another list, indicating the list is shallowly copied. **Important Points:** The difference between shallow and deep copying is only relevant for compound objects (objects that contain other objects, like lists or class instances):

- A shallow copy constructs a new compound object and then (to the extent possible) inserts references into it to the objects found in the original.
- A deep copy constructs a new compound object and then, recursively, inserts copies into it of the objects found in the original.

Example: This code demonstrates shallow copying of a list with nested elements using the '`copy`' module. Initially, it prints the original elements of `li1`, then performs shallow copying into `li2`. Modifying an element in `li2` affects the corresponding element in `li1,` as both lists share references to the inner lists. This illustrates that shallow copying creates a new list, but it doesn't provide complete independence for nested elements.

```
import copy
li1 = [1, 2, [3,5], 4]
```

```
li2 = copy.copy(li1)
print ("The original elements before shallow copying")
for i in range(0,len(li1)):
    print (li1[i],end=" ")

print("\r")
li2[2][0] = 7
print ("The original elements after shallow copying")
for i in range(0,len( li1)):
    print (li1[i],end=" ")
```

Output:

```
The original elements before shallow copying
1 2 [3, 5] 4
The original elements after shallow copying
1 2 [7, 5] 4
```

Источники

- copy in Python (Deep Copy and Shallow Copy)
- Difference between Shallow and Deep copy of a class

Revision #1
Created 15 February 2024 04:39:13 by Антон Сергеевич Абраменко
Updated 15 February 2024 04:43:59 by Антон Сергеевич Абраменко