Shallow Copy & Deep Copy

* Copy an object

To copy an object means creating its exact duplicate in memory.

To copy anything in python we use = , but when we use = to copy in objects it’s doesn’t copy in fact it just makes a new variable points to the object (without making new storage of it), so to copy an object we use either shallow copy or deep copy.

* Shallow copy

A shallow copy makes a new object that have the reference to the original elements So, a shallow copy doesn't create a copy of nested objects (like a list inside a list), instead it just copies the reference of nested objects. We use the copy module of python to create a shallow copy #import copy.

import copy

old\_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

new\_list = copy.copy(old\_list)

print("Old list:", old\_list) #Old list: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

print("New list:", new\_list) #New list: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

If we modify the outer list itself (old\_list[2] = ...), the change does not affect the copied list.  
However, if we modify an element inside a nested object (e.g., old\_list[2][2] = ...), the change will also appear in the copied list because both lists share the same nested object in memory.

* Deep copy

A deep copy makes a completely new object, and also makes new copies of all the objects inside it, so nothing is shared with the original. So even if we edit a nested element in the original object is does NOT change the deep copied object

# new\_list = copy.deepcopy(old\_list)