## Deep copy & shallow copy :-

For normal or single dimensional list shallow copy and deep copy are same.

But in nested list, both are different that if we change in list 1 [1] [0] element than the list 2 [1] [0] didn't get changed.

- In single dimensional list or normal list when we copy using 'copy.deepcopy()' the element didn't get changed

```
list 1 = [1, 2, 3, 4]
list 2 = copy.deepcopy( list 1)

list 1 [2] = 100
list 1
o/p [1, 2, 100, 4]

list 2
o/p [1, 2, 3, 4]   ∴ element at index value [2] didn't get changed.
```

Example:

```
# deepcopy shallow copy

list 1 = [1, 2, 3, 4]
list 2 = list 1
list 1          [o/p : 1, 2, 3, 4]
list 2          [o/p : 1, 2, 3, 4]
```

# assign new element at index 1 of list 2

list2[1] = 1000
list 2 [O/P:- 1, 1000, 3, 4]

list 1 [O/P:- 1, 2, 3, 4]

id (list1) [O/P:- 2310184965184]
id (list2) [O/P:- " ]

↓

[In this case both list memory allocation is same & the reference id of each element is also same.]

# Copy operator
# shallow copy

list 1 = [1, 2, 3, 4]
list 2 = list1.copy()

list 1 [O/P: 1, 2, 3, 4]
list 2 [O/P " ]

list 2[1] [O/P: 2]
list 1[1] [O/P: 2]

list 2[1] = 1000
list 2 [O/P: 1, 1000, 3, 4]
list 1 [O/P: 1, 2, 3, 4]

↓

[In this the memory allocation of each list is different and the reference of each element is also difference]

id ( list 1) [o/p:— 2310| 84965184 ]
id (list 2] [o/p:— 2310| 56 96518 2] > Both are differ.

Shallow copy with nested list :-

list 1 = [[1,2,3,4],[5,6,7,8]]
list 2 = list 1. copy ( )

list 1 [op:[1,2,3,4], [5,6,7,8]]

list 2 [op " " ]

list 1[1][0] [op:5]
list 1[1][0] = 100

list 1 [o/p :[[1,2,3,4],[100,6,7,8]]
list 2 [o/p:[[ " ], [ " ]]
↳ [In the nested list the memmory allocas of both list is difference, but the reference of each element of ~~index~~ each index are same they both work on the same reference id ]

list 1. append ([2,3,4,5])
list 1 [o/p:[[1,2,3,4],[100,6,7,8],
[2,3,4,5]]

list 2 [o/p: [[1,2,3,4],[100,6,7,8]]
↳
In this case the memory allocation is different but the index value and their reference are also different
Ex: list [2] [3]
↳ This is different.

# deep copy

import copy [for useing deepcopy we have to import library copy]

list 1 = [1, 2, 3, 4]
list 2 = copy.deepcopy(list 1)

list 2[1] [o/p:- 2]

list 2[1] = 100

list 2 [o/p:- 1, 100, 3, 4]

list 1 [o/p - 1, 2, 3, 4]

id (list 1) [o/p :- 231440671]
id (list 2) [o/p :- 231430576]

## in a normal copy shallow copy == deepcopy

list 1 = [[1, 2, 3], [3, 4, 5], [5, 6, 7]]
list 2 = copy.deepcopy (list 1)

list 1[1][0] [op: 3]

list 2 [1][0] [op : 100]

list 2 [o/p:[[1, 2, 3], [100, 4, 5], [5, 6, 7]]

id (list 1) [op: address]

id (" 2) [op: address differ]

> To find reference id of each element of double dimensional list/array :-

l = [[1,2,3],[4,5,6]]

id(l) [OP : 194732 ---- (address)]

id(l[1][0]) [OP: 140707 84593608 (address)]

b = l.copy()

b [OP: [[1,2,3],[4,5,6]]]

id(b) [OP: 194732 261 86944 (address diffe from 'l' address)]

id(b[1][0]) [OP:- 1947078 4 ---- (address)]

id(b

id(b[1]) [OP:- 194732 661 8688]

id(l[1]) [OP:- '' (Same as id of 'b[1]' because it is double dimensional array)]

id(l[1][0]) [OP:- same address as id of 'b[1][0]']

> Single dimensional list :-

x = [1,2,3]

y = x.copy()

id (x) [OP - address]

id (y) [OP - " (differs from id of 'x')]

It means copy of array is created at different place.

id (x[1]) [OP :- address]

id (y[1]) [OP :- " (same as id of y[1])

y[1] = 5

x [OP :- [1, 2, 3]]

y [OP :- [1, 5, 3]]

id (x[1]) [OP :- address]

id (y[1]) [OP :- " (differ from address of (x[1]))

id [OP :- [1, 2, 3] [4, 5, 6]]

b[1][0] = 9

b [OP :- [1, 2, 3] [9, 5, 6]]

l [OP :- ]

id (b) [OP :- address] ⟷ Same

id (l) [OP :- address]

id (b[1][0]) [OP :- address] ⟷ both are same.

id (b[1][0]) [OP :- " ]