Python (Arrays & list)

List - store multiple elements at same time of different type.

n size → list // Heterogeneous

indexing → 0 to n-1 // stored continuously

'[ ]' → list

Accessing elements of a list → li[index_number]

li[0], li[2]...

Similarily, for changing element → li[index_number] = value

li[0]=3 , li[4]='5'

Note: Must be within the index range

Slicing of list: generates a part of the list.

li[1:3] => ~~→2~~ index elements at 1 & 2

li[1:] => 1 to len(li-1), till end

li[:] => entire list

li[1:100] => 1 to end element doesn't matter if index is out of range.

Insert & appending elements in list:

li.append(value) → adds element to last

li.insert(index_number, value) → adds value at particular index

↳ even if index out of range, inserts at last position li.append([val1, val2, val3...]) → adds the given list to already existing list as list itself.

li = [1, 2 ..., 8, [val1, val2 ...]]

Adding multiple elements to list → extend(val1, val2, ...)

li.extend(val1, val2, ...)

adds the elements as values (not list) into existing list.

Removing elements from list -

li.remove(particular_value)

if multiple value present, first index incoming of that value will be removed.

li = [5, 6, 7 ... 5]

li.remove(5)

li = [6, 7 ... 5]

li.pop(index_value) - removes element from particular index & returns the deleted element.

By default, removes from end.

Elements storing in list -

Reference of each element is stored.

Reference

li = [1, 2, 3.4, 'Prakash'] => [100, 200, 300, 400]

0 1 2 3

Index = 0 1 2 3

li[0] → Reference (100) $\xrightarrow{\text{returns}}$ element ~~so~~

1 3 6 2 5 4 3 2 4

(2,5) (6,1) (4,3) ~~(4,5)~~
(5,2) (1,6) (3,4)
(4,3)

⑫

1 3 6 2 5 4 3 2 4

When elements changed,

li[1] = 77

750

77 → $\boxed{77}$

new reference of list => [100, ~~200~~ 750, 300, 400]

// References are stored continuously → meaning 100, 104, 108, 112 (if 4 bytes)

Resizing of list

When we add new element, ~~python creates~~ to an existing list,
python creates a new list (double the size - not in all cases)
then ① copies old list elements to new list (copies reference)
② and adds the new element at the end (append() case)

Thus, maintaining continuity.

li = [1, 2, 3]

li = [10, 2, 3]

100	104	108	Reference

li.append('Hello')

li = [1, 2, 3, ~~4~~ 'Hello']

0	1	2	3	4		7	
100	104 ~~252~~	108 ~~254~~	112 ~~256~~	116 ~~258~~	. . .	7	Reference

li.append('8') → just adds it to new list

Negative indexing in list

li[-1] → last element

li[-n]
→ must be within index range

Sequencing in list

start

list[: :] ⟹ generates a part of the list.

→ step value

→ end

→ but it only considers until end - 1

li[start :] ⟹ from start index to ~~las~~ end of the list (len(li))

By default~~:~~ step = 1, end = len(li)

li[: end] ⟹ from 0th element to end' index.

Default :- start = 0, step = +1

li[-start :] ⟹ ~~returns last element start to end~~. returns last element to ~~at~~ end.

~~By~~ Default : end = len(li), step = 1

Space separated i/p

split( ) → delimiter → anything which will separate elements by

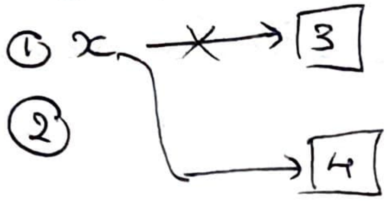
default → delimiter - ' ' (space) delimiter

Mutable & immutable concept

Variables → immutable

① x = 3

x = 4



// reference changed.

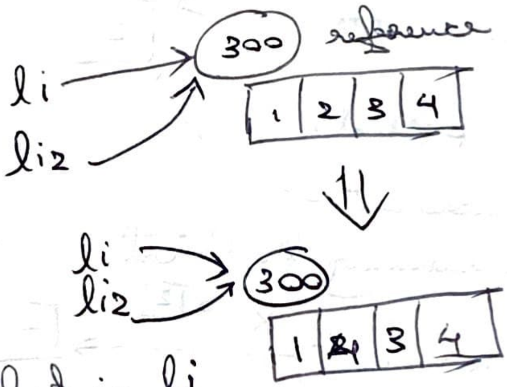
List → mutable.

li = [1, 2, 3, 4]

li = $\boxed{300}$

li2 = li

li2[1] = 4



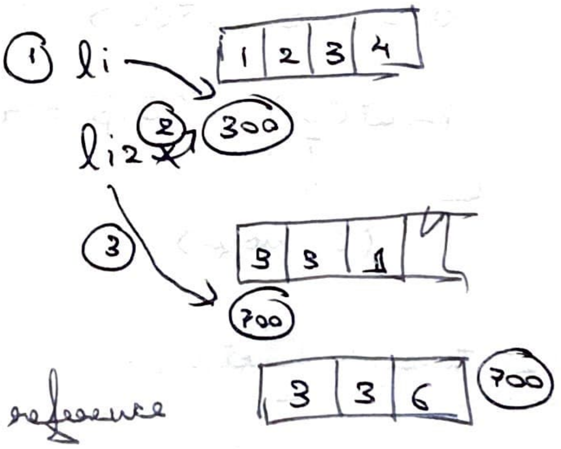
Changes in li2 will reflect in li also. Since, both are referring to same reference.

① li = [1, 2, 3, 4]

② li2 = li

③ li2 = [3, 3, 1]

④ li2 [2] = [6]



No changes to li, since reference changed.