Thanks for sharing the screenshot! You're demonstrating a very important concept in Python: **reference vs copy** for mutable objects like **lists**.



```
✓ Case 1: 11 = [1, 2, 3], 12 = 11
```

```
python

11 = [1, 2, 3] 12 = 11
```

- You did **not** create a new list for 12.
- Both 11 and 12 point to the same list in memory.

Then you did:

This modifies the original list, so now:

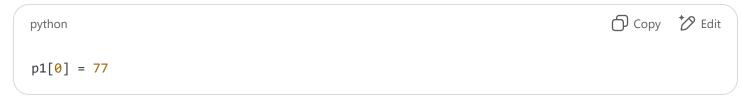
python
$$\bigcirc$$
 Copy $^{\mbox{to}}$ Edit $11 \rightarrow [77, 2, 3] \ 12 \rightarrow [77, 2, 3] \ \checkmark$ (because they are the same object)

```
python

p1 = [1, 2, 3] p2 = [1, 2, 3]
```

Here, you created **two different lists** with the same values. So p1 and p2 point to **different objects**.

Then you changed:



Now:



Summary:

| Statement | Points to Same Object? | Changes Affect Both? |
|----------------|------------------------|----------------------|
| 12 = 11 | ✓ Yes | ✓ Yes |
| p2 = [1, 2, 3] | X No | X No |

Wey Concept:

- 12 = 11 is a reference assignment.
- p2 = [1, 2, 3] is a new list with same content.
- To make a **copy** of a list (not just a reference), use:

```
python

new_list = old_list.copy() # or import copy new_list = copy.deepcopy(old_list) # for
nested lists
```

NOTE:

- number above is an immutable so cannot be change so refrence change hudaina.
- as list is mutable so refrence can be change so simply can be understand from above figure.