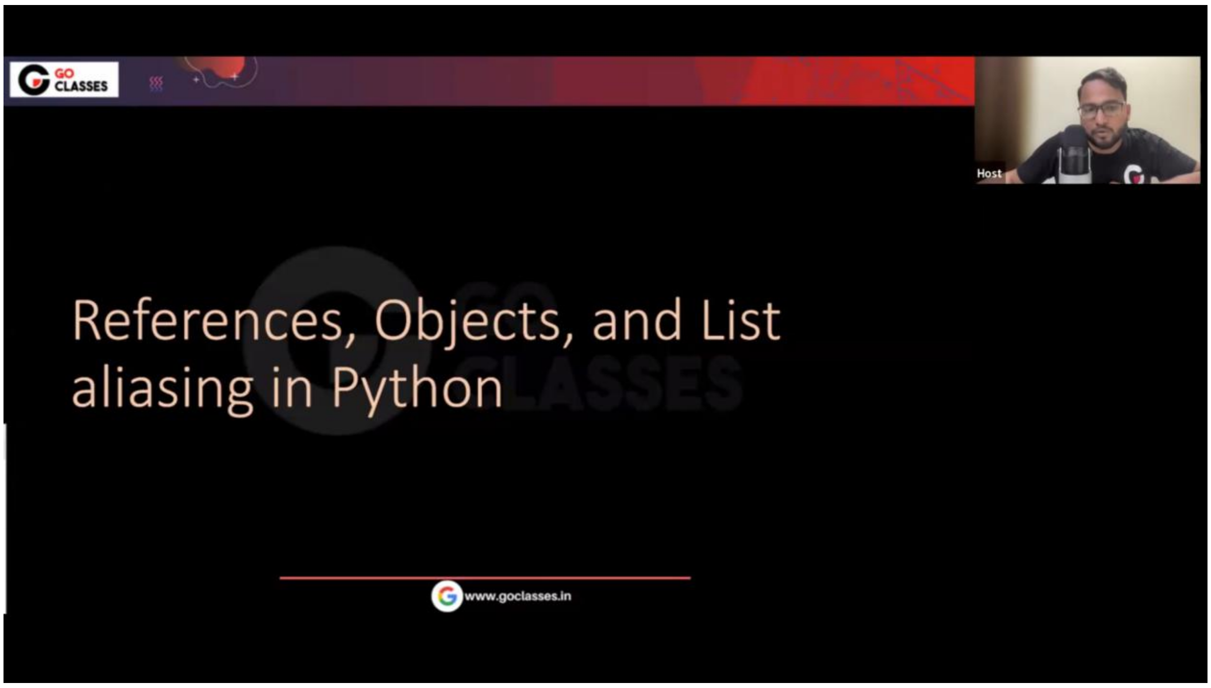
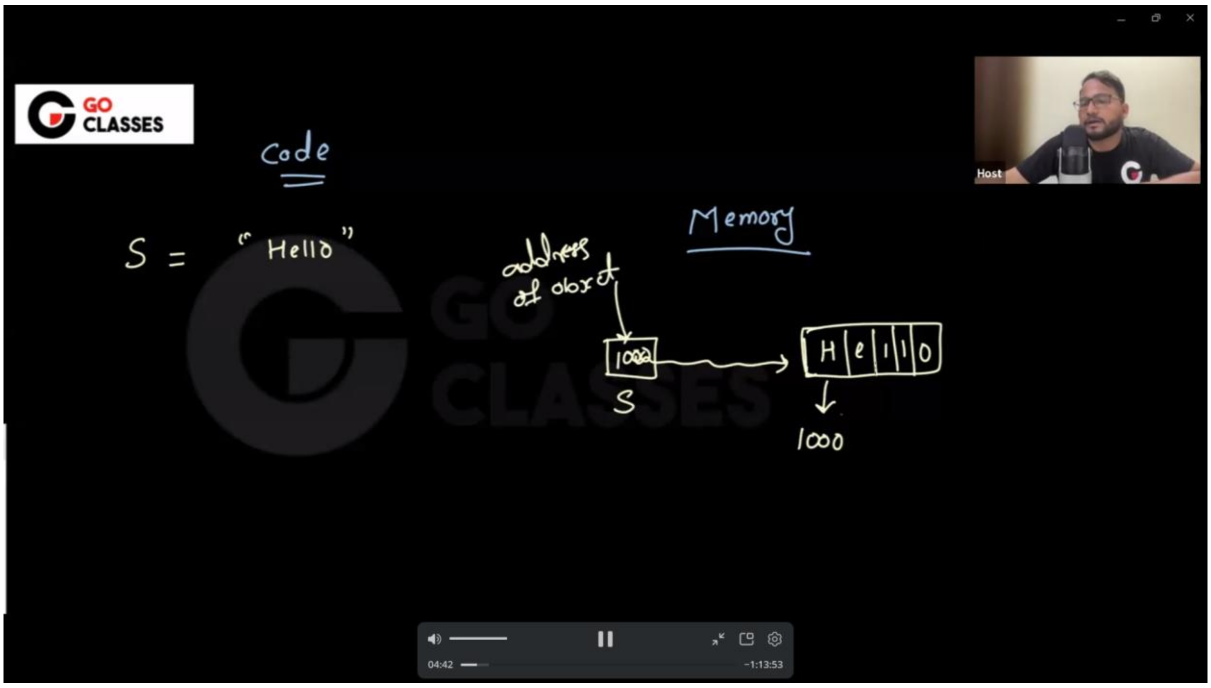
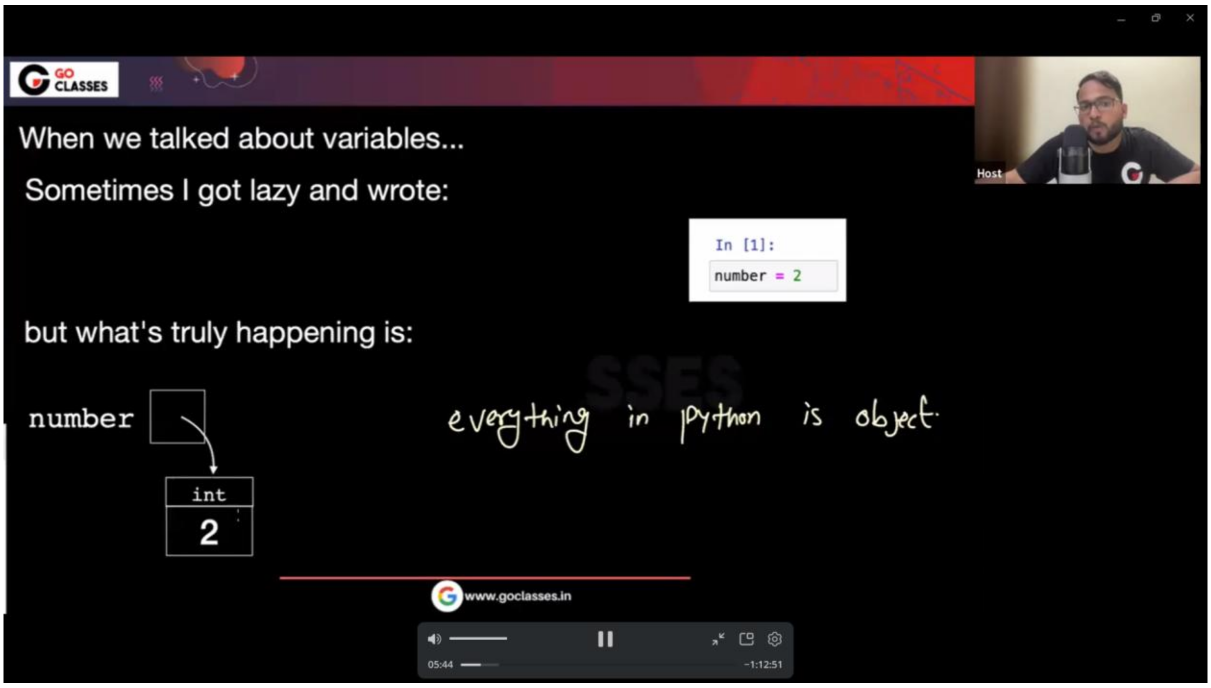


# References, Objects, and List aliasing in Python



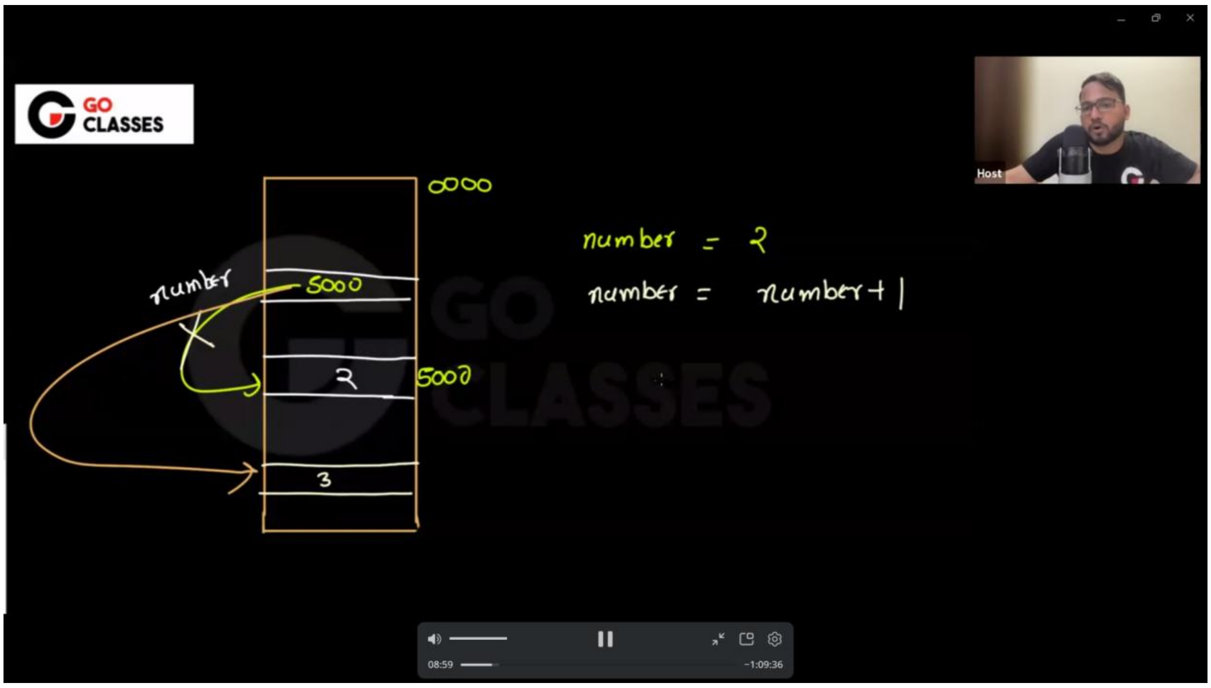


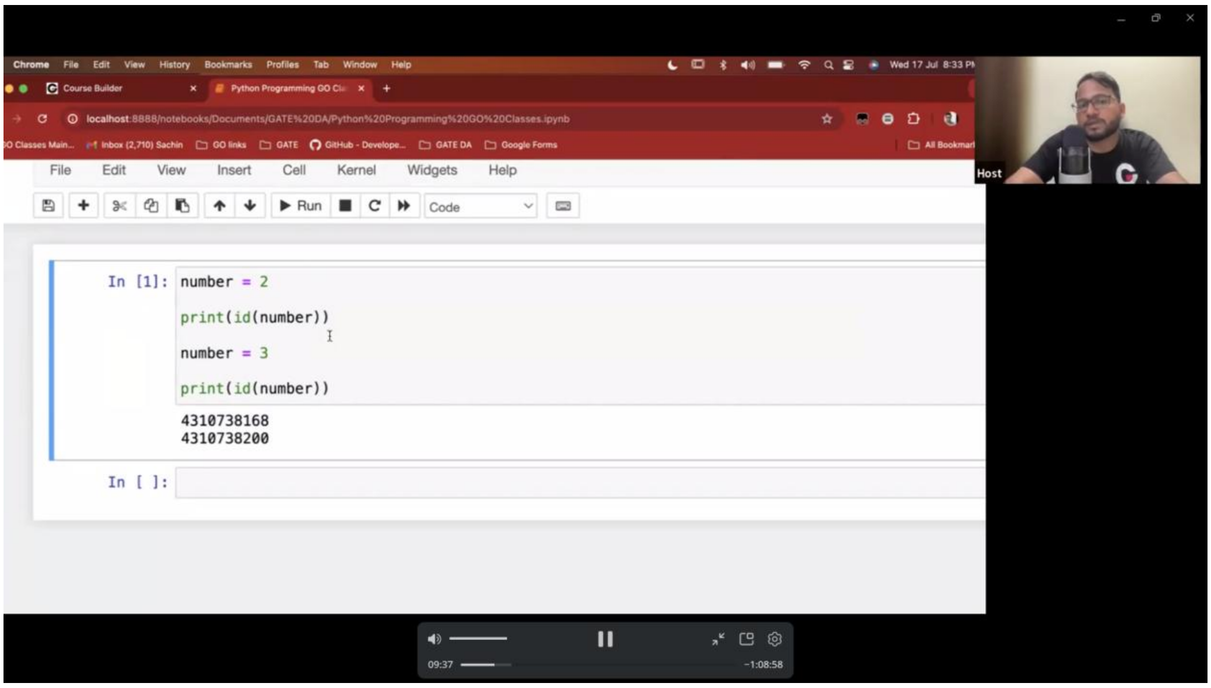
When we talked about variables...

Sometimes I got lazy and wrote:

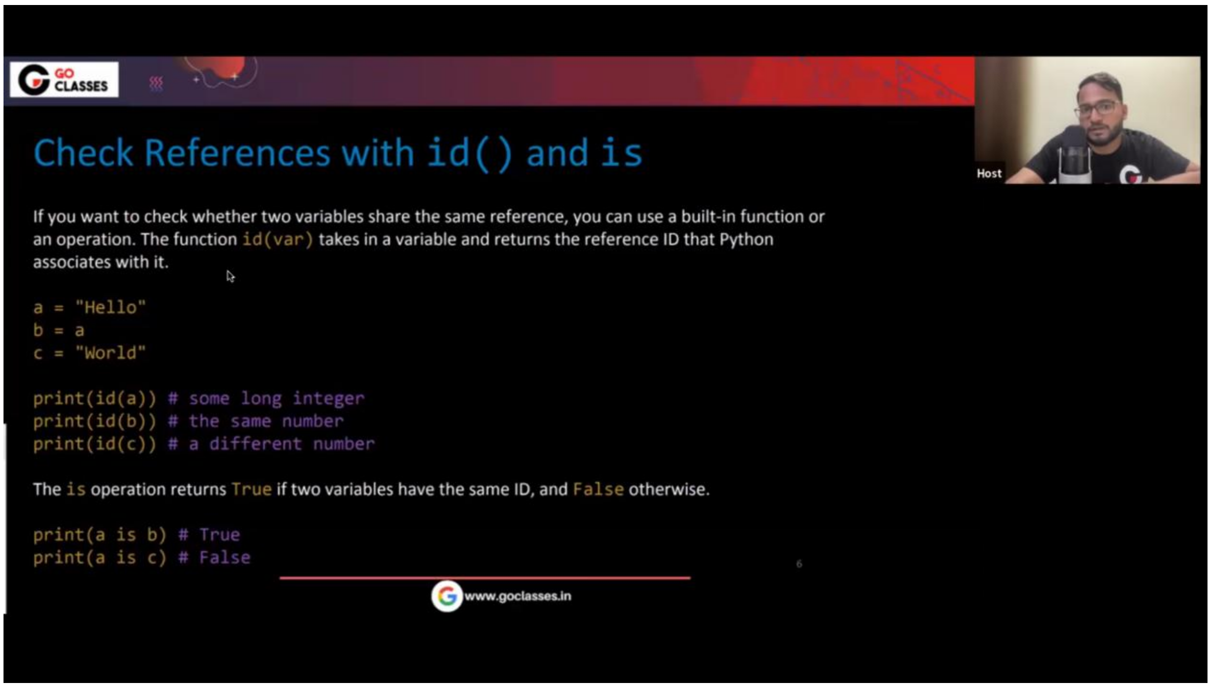
```
In [1]:
number = 2
```

but what's truly happening is:




Chrome File Edit View History Bookmarks Profiles Tab Window Help
Wed 17 Jul 8:33 PM
Course Builder
Python Programming GO Cla
localhost:8888/notebooks/Documents/GATE%20DA/Python%20Programming%20GO%20Classes.ipynb
File Edit View Insert Cell Kernel Widgets Help
Run
Code
Host
09:37
-1:08:58

```
In [1]: number = 2

        print(id(number))

        number = 3

        print(id(number))

        4310738168
        4310738200

In [ ]:
```



## Check References with id() and is

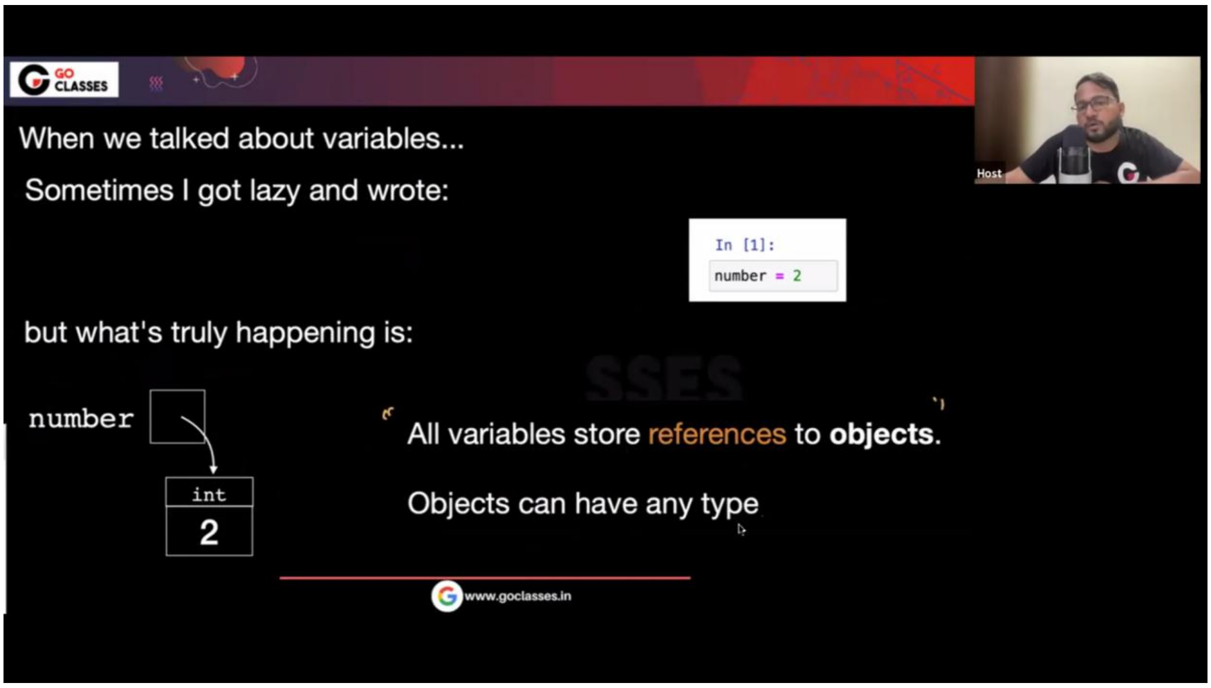
If you want to check whether two variables share the same reference, you can use a built-in function or an operation. The function id(var) takes in a variable and returns the reference ID that Python associates with it.

```
a = "Hello"
b = a
c = "World"

print(id(a)) # some long integer
print(id(b)) # the same number
print(id(c)) # a different number
```

The is operation returns True if two variables have the same ID, and False otherwise.

```
print(a is b) # True
print(a is c) # False
```

6



When we talked about variables...

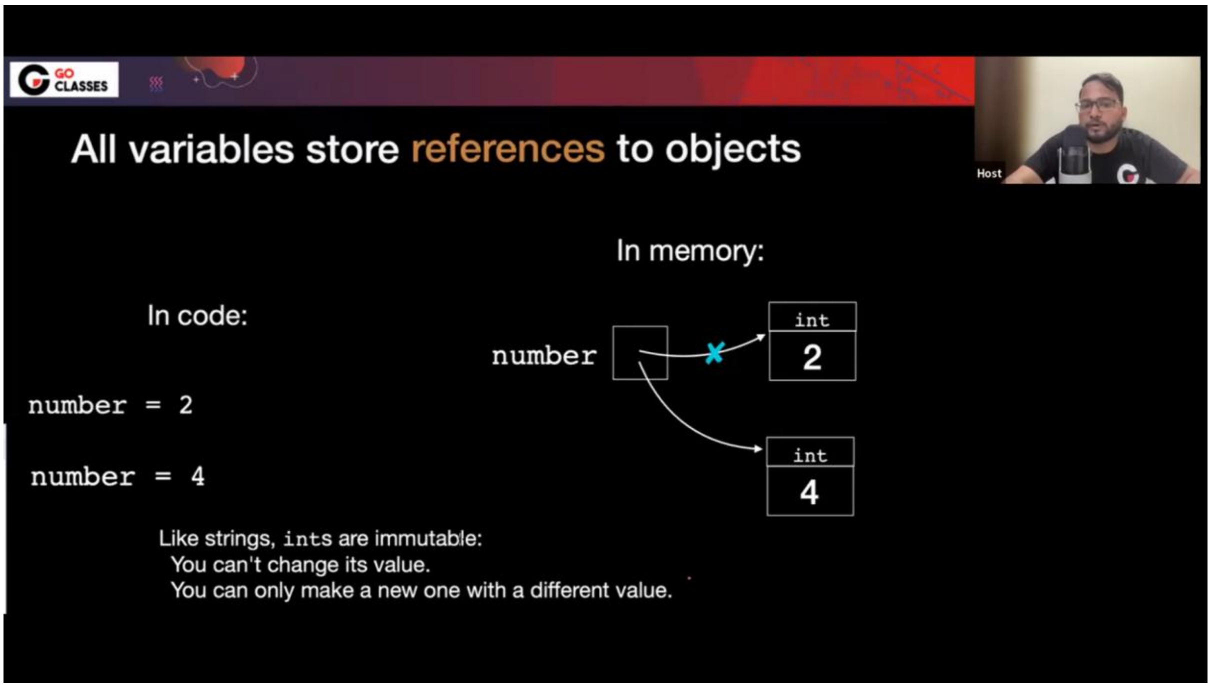
Sometimes I got lazy and wrote:

```
In [1]:
number = 2
```

but what's truly happening is:

All variables store references to **objects**.

Objects can have any type



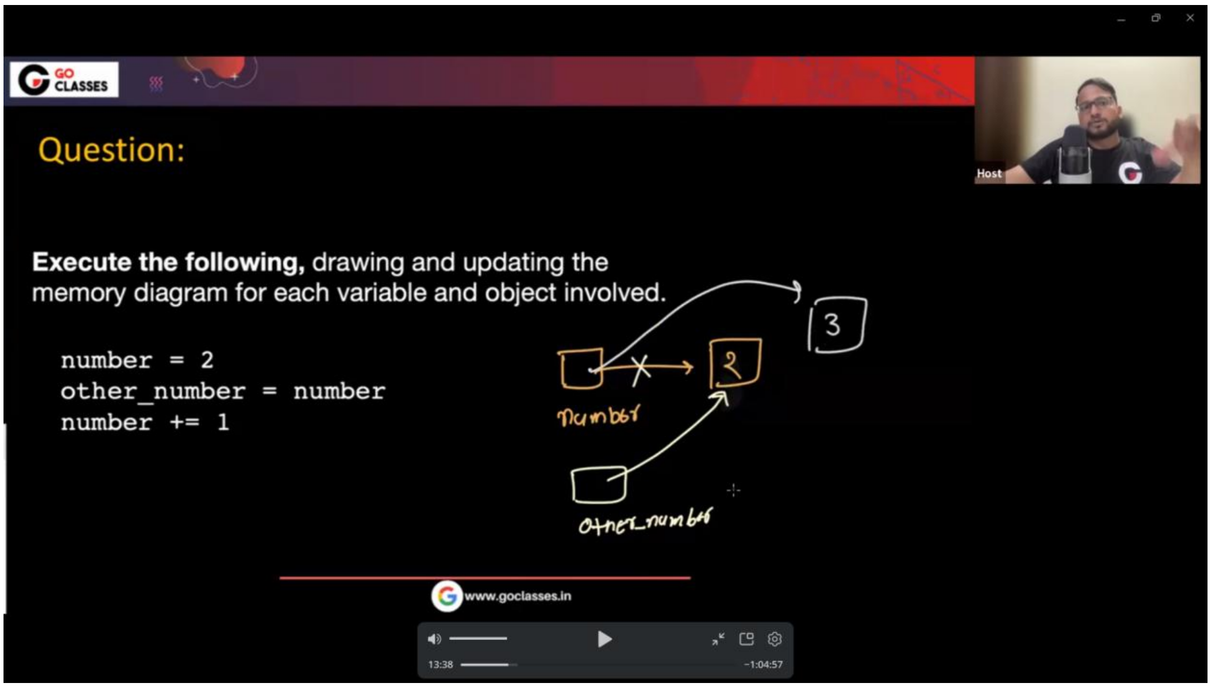
## All variables store references to objects

In code:

```
number = 2

number = 4
```

In memory:

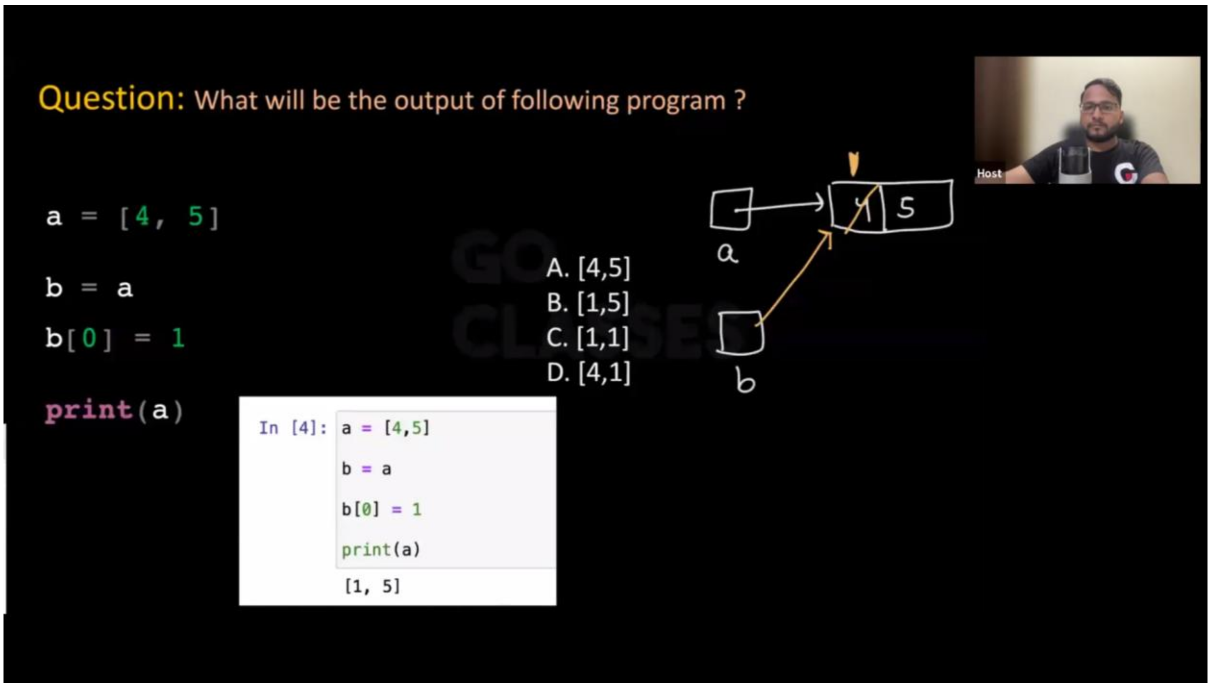
Like strings, `int`s are immutable:
You can't change its value.
You can only make a new one with a different value.



## Question:

**Execute the following,** drawing and updating the memory diagram for each variable and object involved.

```
number = 2
other_number = number
number += 1
```

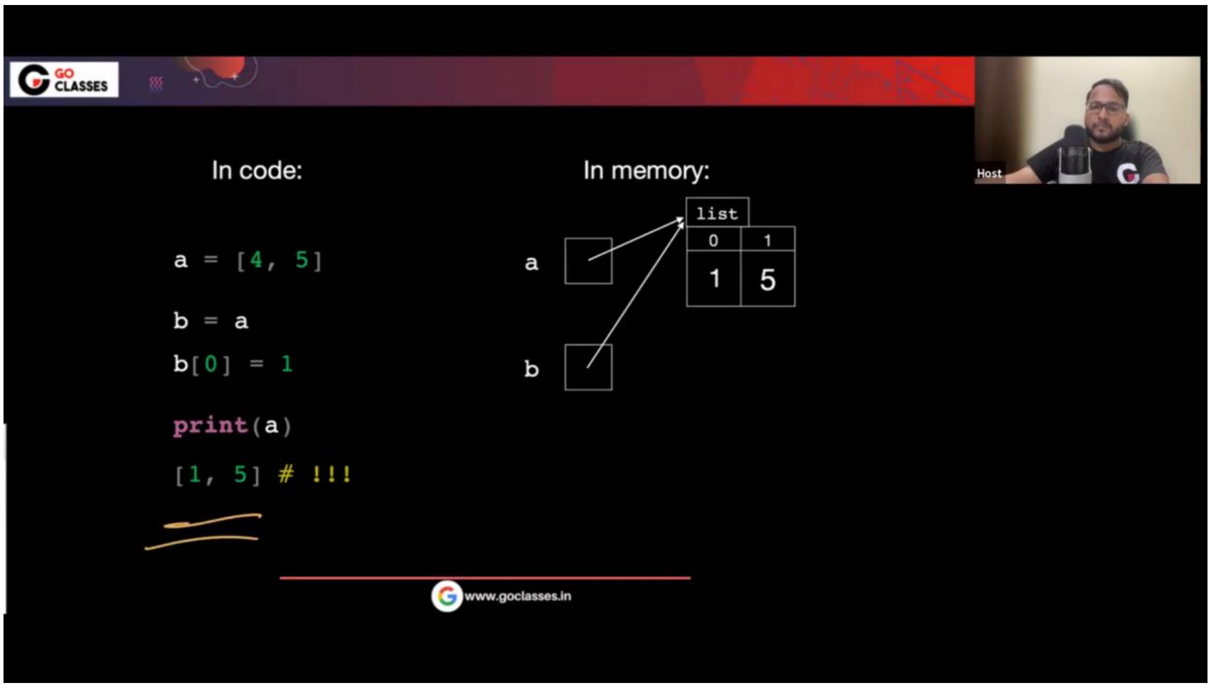


## Question: What will be the output of following program ?

```
a = [4, 5]

b = a

b[0] = 1

print(a)
```

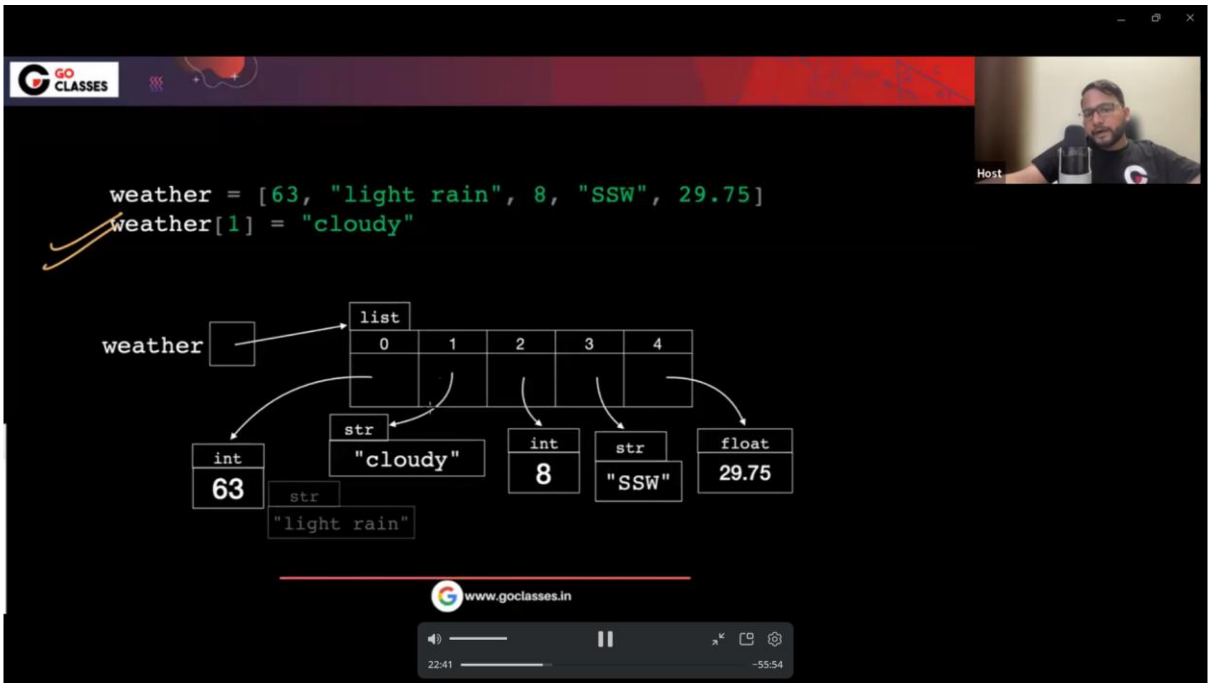
A. [4,5]
B. [1,5]
C. [1,1]
D. [4,1]

```
In [4]: a = [4,5]

        b = a

        b[0] = 1

        print(a)

        [1, 5]
```

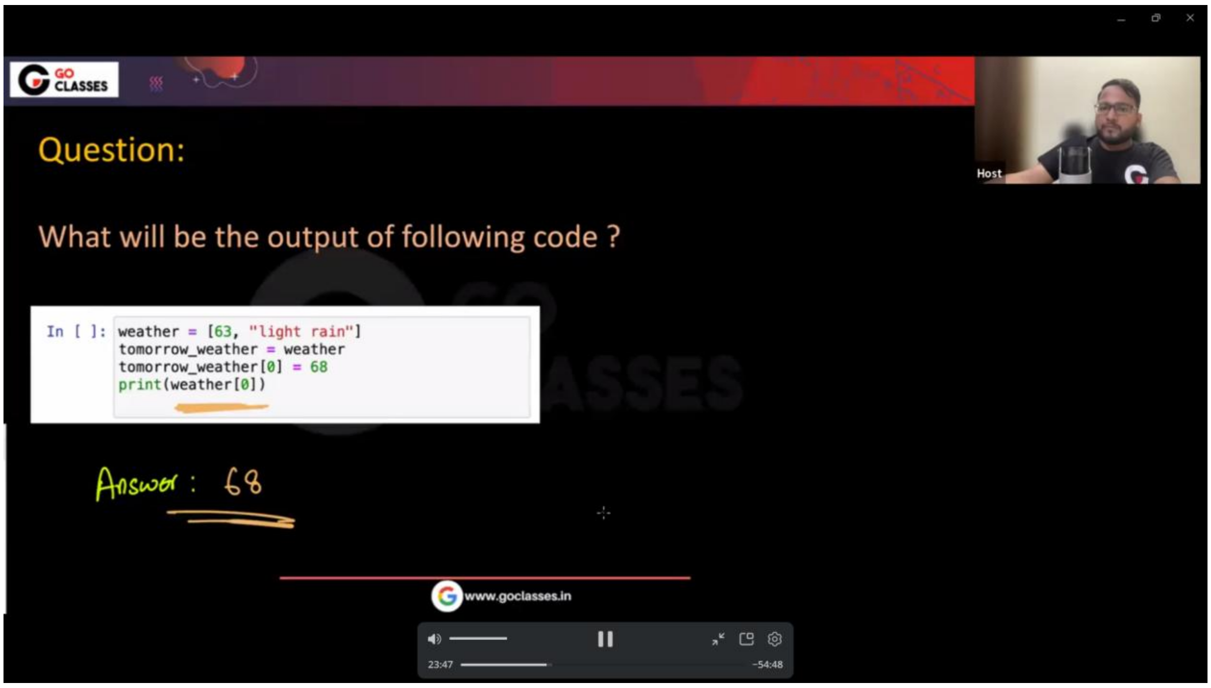

In code:

```
a = [4, 5]

b = a
b[0] = 1

print(a)
[1, 5] # !!!
```

In memory:



```
weather = [63, "light rain", 8, "SSW", 29.75]
weather[1] = "cloudy"
```

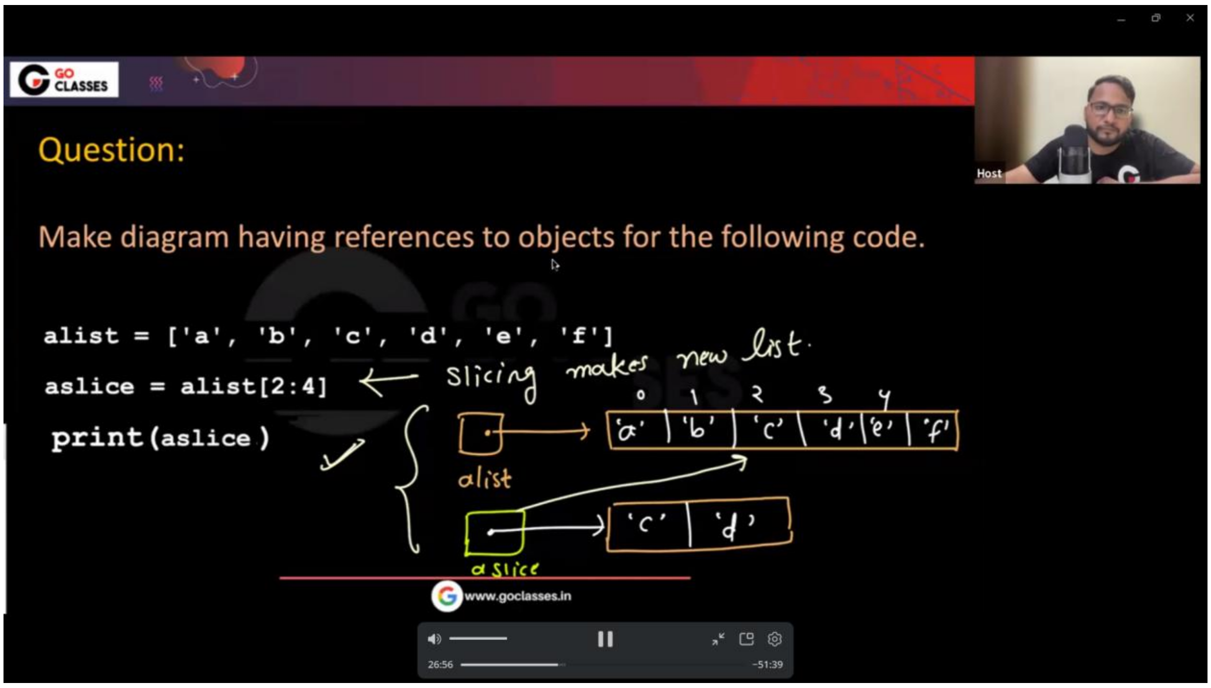


## Question:

What will be the output of following code ?

```
In [ ]: weather = [63, "light rain"]
        tomorrow_weather = weather
        tomorrow_weather[0] = 68
        print(weather[0])
```

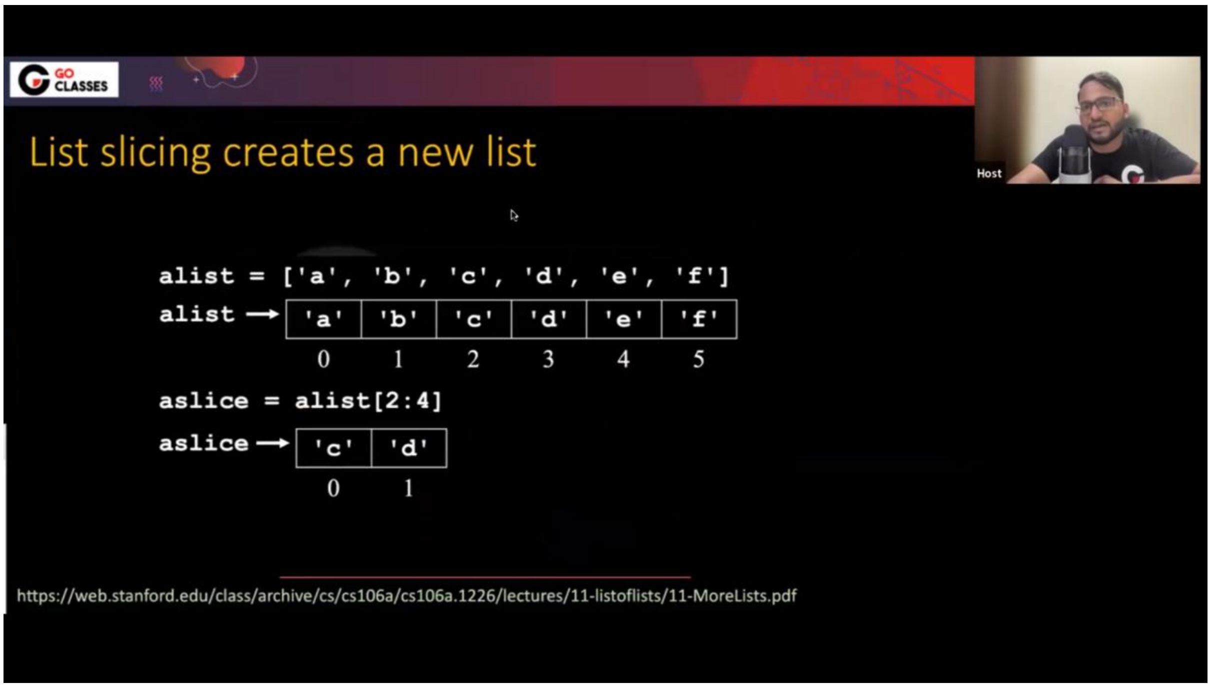
Answer : 68



## Question:

Make diagram having references to objects for the following code.

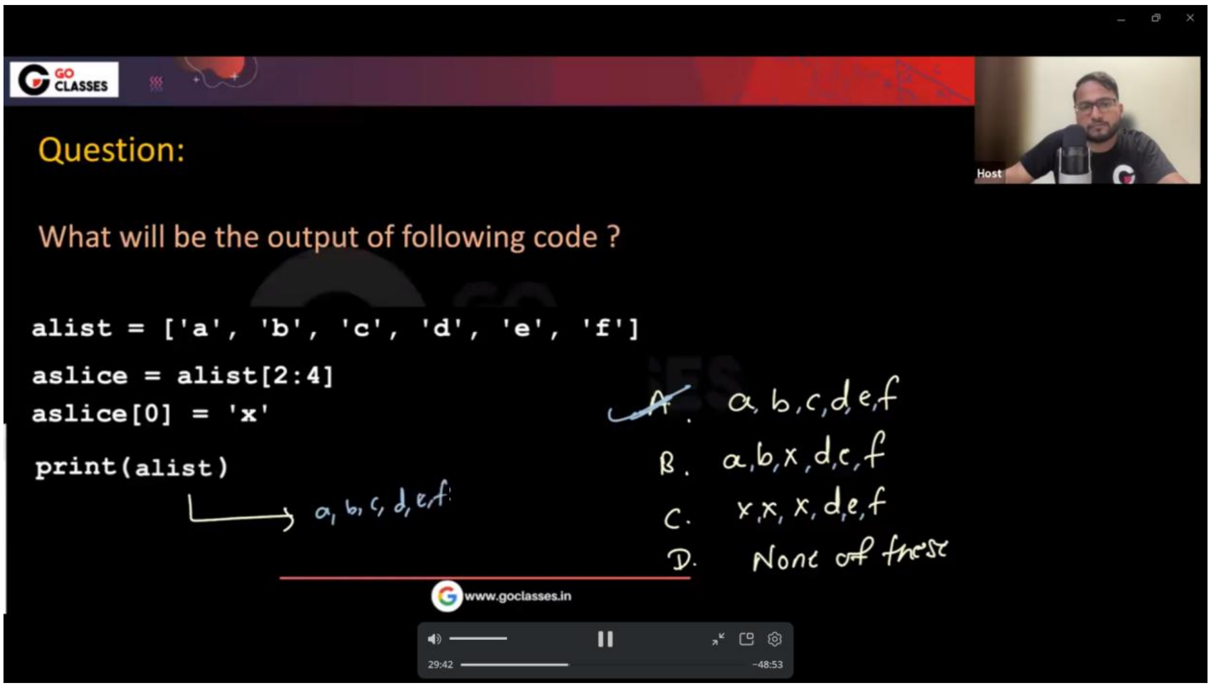
```
alist = ['a', 'b', 'c', 'd', 'e', 'f']
aslice = alist[2:4]
print(aslice )
```



## List slicing creates a new list

```
alist = ['a', 'b', 'c', 'd', 'e', 'f']
alist → | 'a' | 'b' | 'c' | 'd' | 'e' | 'f' |
           0     1     2     3     4     5
aslice = alist[2:4]
aslice → | 'c' | 'd' |
            0     1
```

https://web.stanford.edu/class/archive/cs/cs106a/cs106a.1226/lectures/11-listoflists/11-MoreLists.pdf



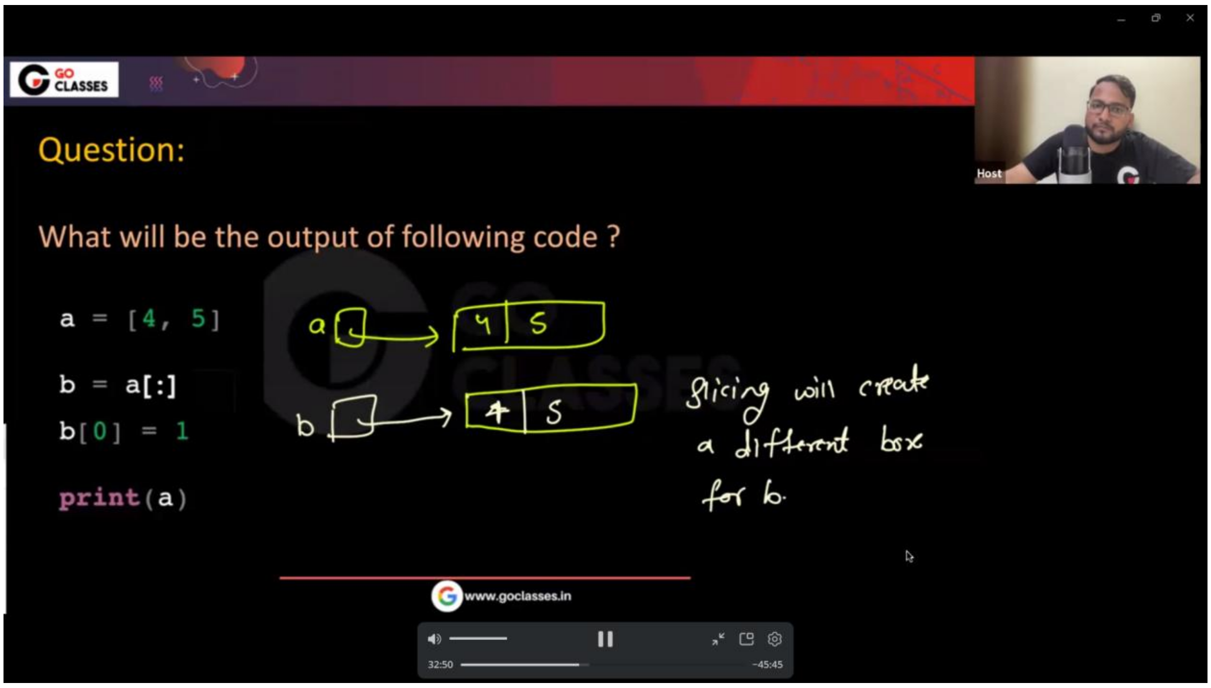
## Question:

What will be the output of following code ?

```
alist = ['a', 'b', 'c', 'd', 'e', 'f']
aslice = alist[2:4]
aslice[0] = 'x'
print(alist)
```

$\longrightarrow$ a, b, c, d, e, f

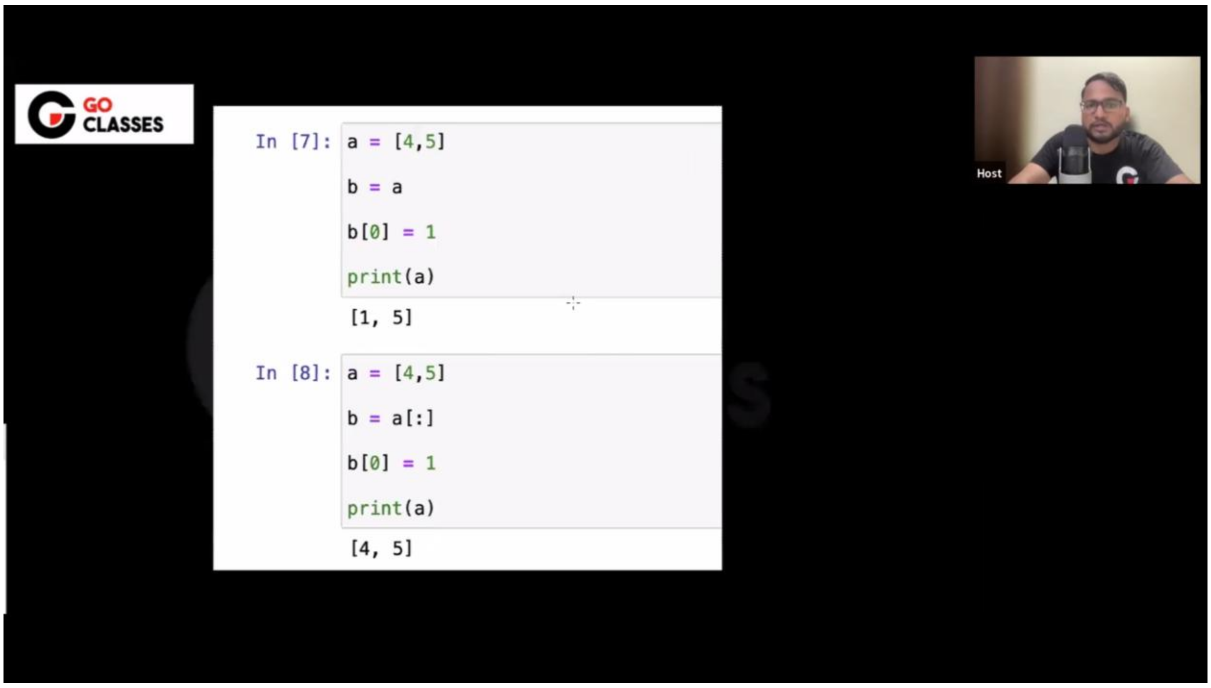
[✓ A. a, b, c, d, e, f]
B. a, b, x, d, e, f
C. x, x, x, d, e, f
D. None of these



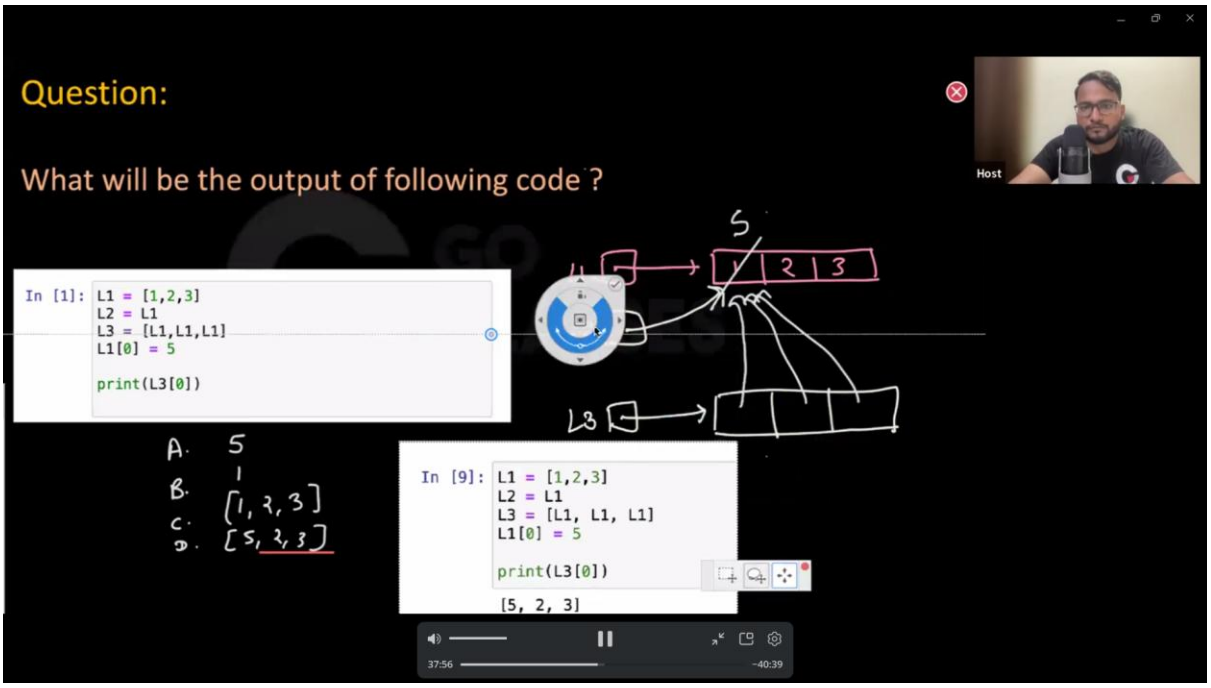
## Question:

What will be the output of following code ?

```
a = [4, 5]

b = a[:]
b[0] = 1

print(a)
```



```
In [7]: a = [4,5]

        b = a

        b[0] = 1

        print(a)

        [1, 5]

In [8]: a = [4,5]

        b = a[:]

        b[0] = 1

        print(a)

        [4, 5]
```



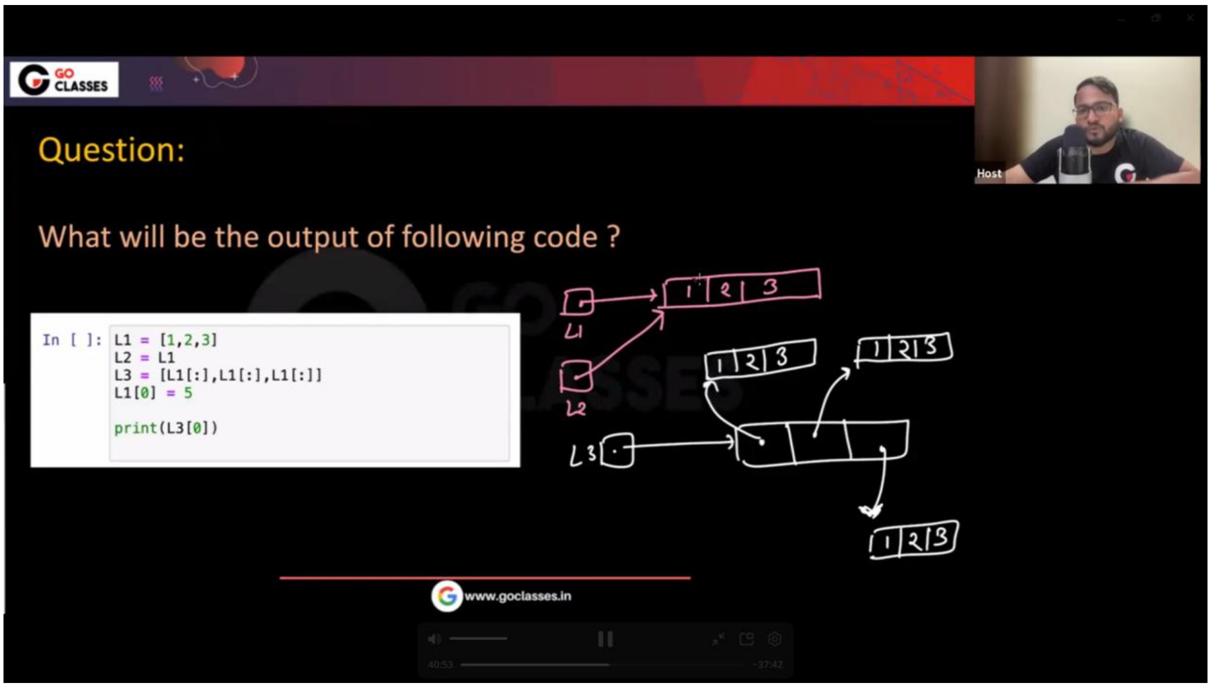
## Question:

What will be the output of following code ?

```
In [1]: L1 = [1,2,3]
        L2 = L1
        L3 = [L1,L1,L1]
        L1[0] = 5

        print(L3[0])
```

A. 5
B. 1
C. [1, 2, 3]
D. [5, 2, 3]

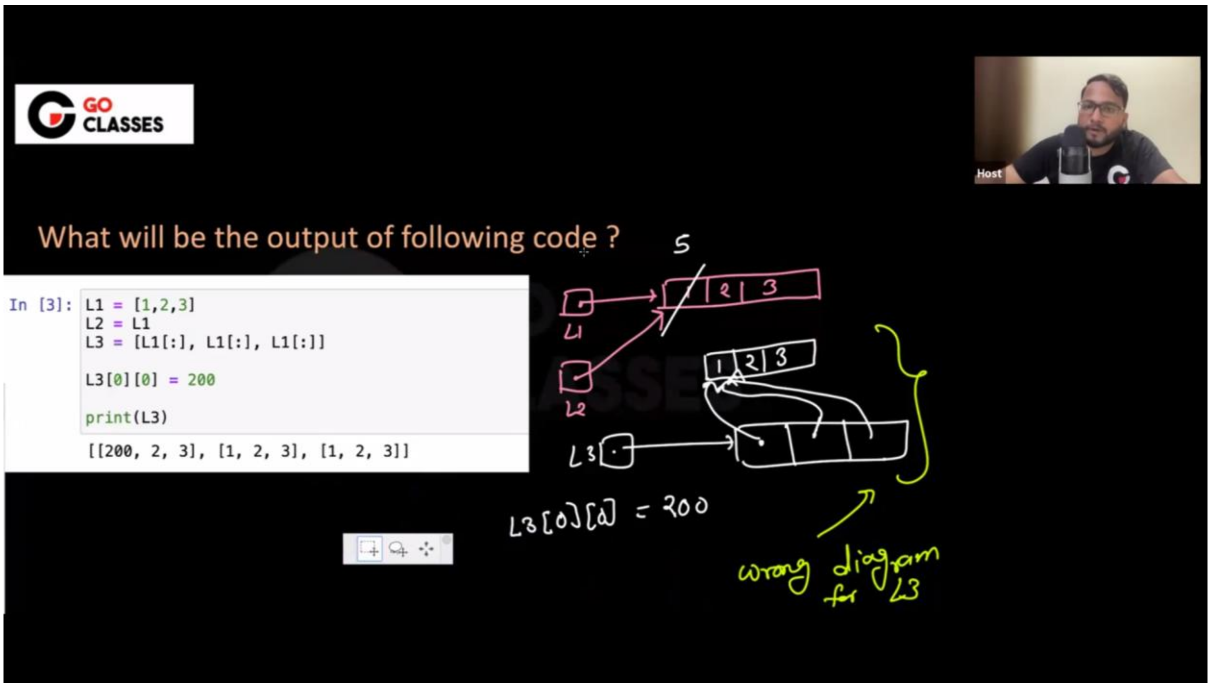
```
In [9]: L1 = [1,2,3]
        L2 = L1
        L3 = [L1, L1, L1]
        L1[0] = 5

        print(L3[0])

        [5, 2, 3]
```



## Question:

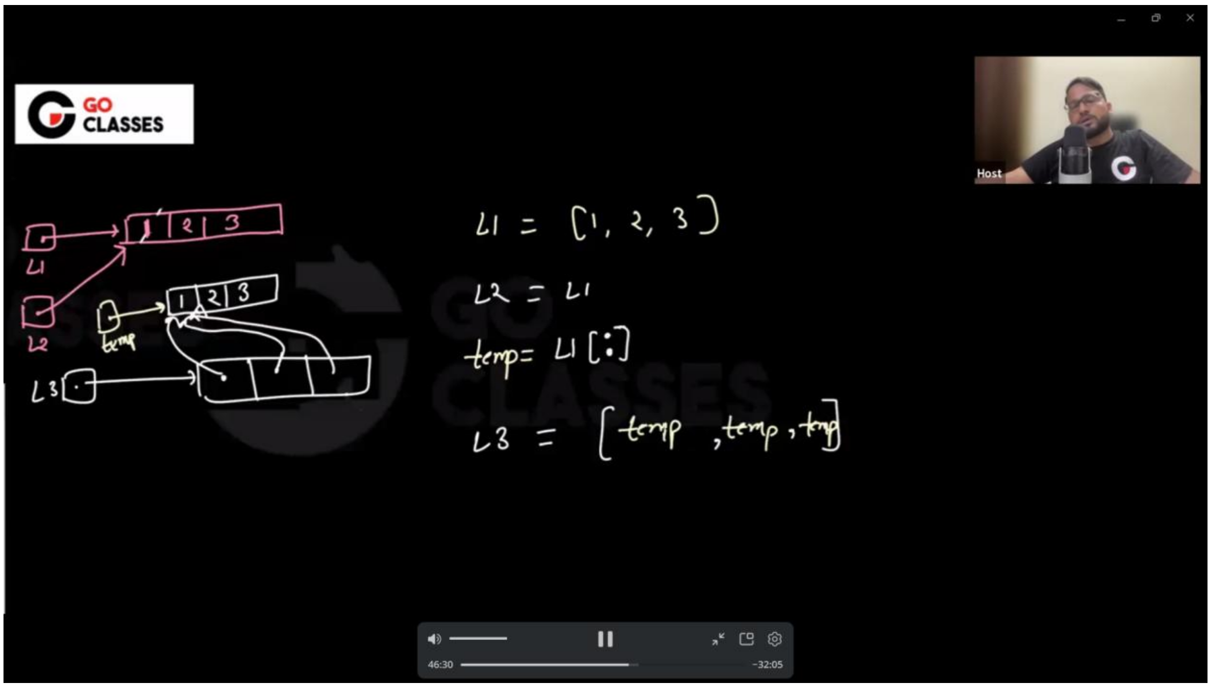
What will be the output of following code ?

```
In [ ]: L1 = [1,2,3]
        L2 = L1
        L3 = [L1[:],L1[:],L1[:]]
        L1[0] = 5

        print(L3[0])
```



What will be the output of following code ?

```
In [3]: L1 = [1,2,3]
        L2 = L1
        L3 = [L1[:], L1[:], L1[:]]

        L3[0][0] = 200

        print(L3)

        [[200, 2, 3], [1, 2, 3], [1, 2, 3]]
```

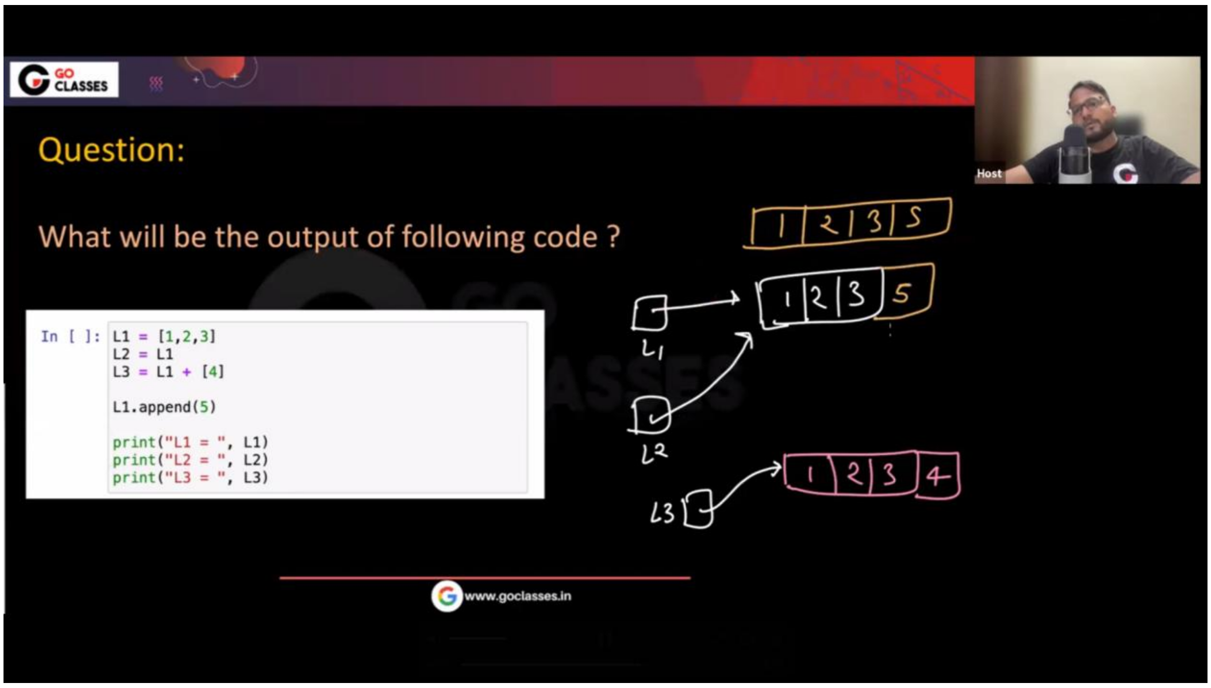


L1 = [1, 2, 3]

L2 = L1

temp = L1[:]

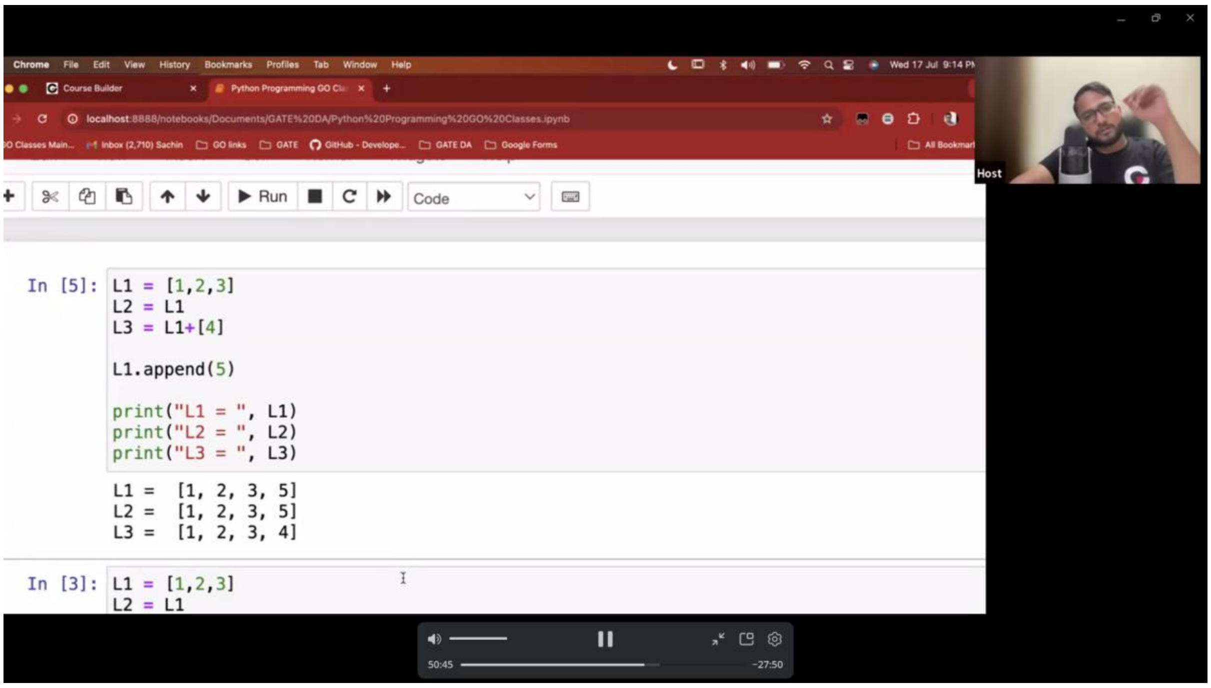
L3 = [temp , temp , temp]



## Question:

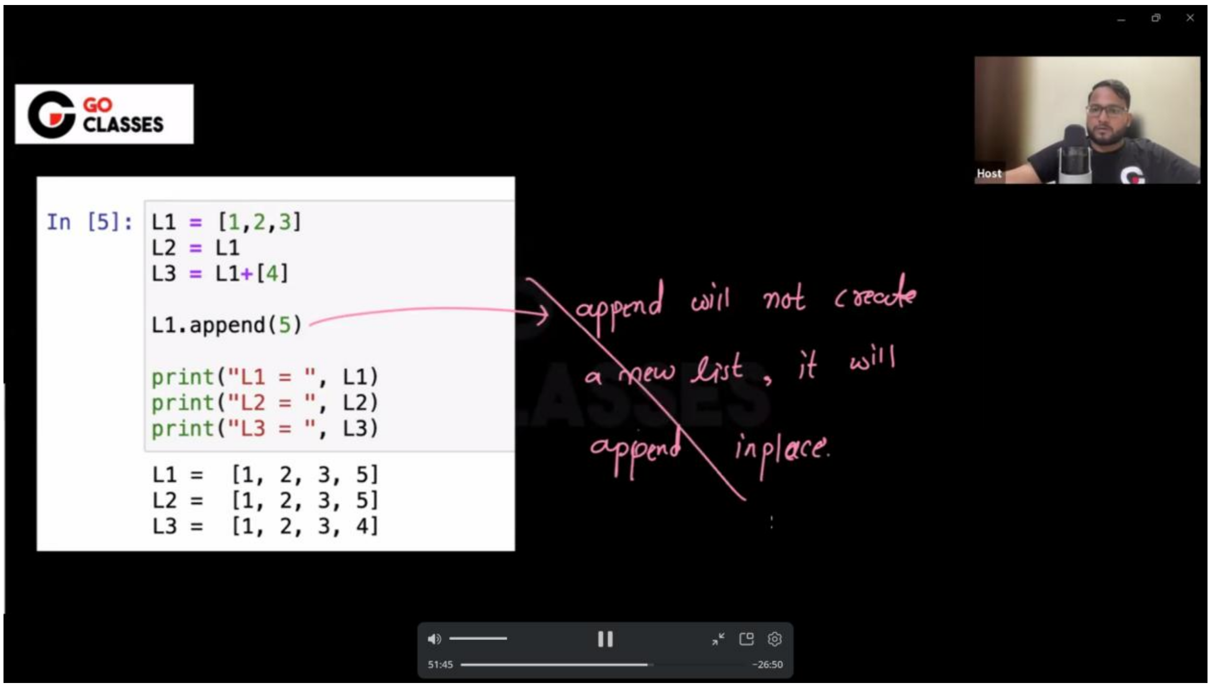
What will be the output of following code ?

```
In [ ]: L1 = [1,2,3]
        L2 = L1
        L3 = L1 + [4]

        L1.append(5)

        print("L1 = ", L1)
        print("L2 = ", L2)
        print("L3 = ", L3)
```



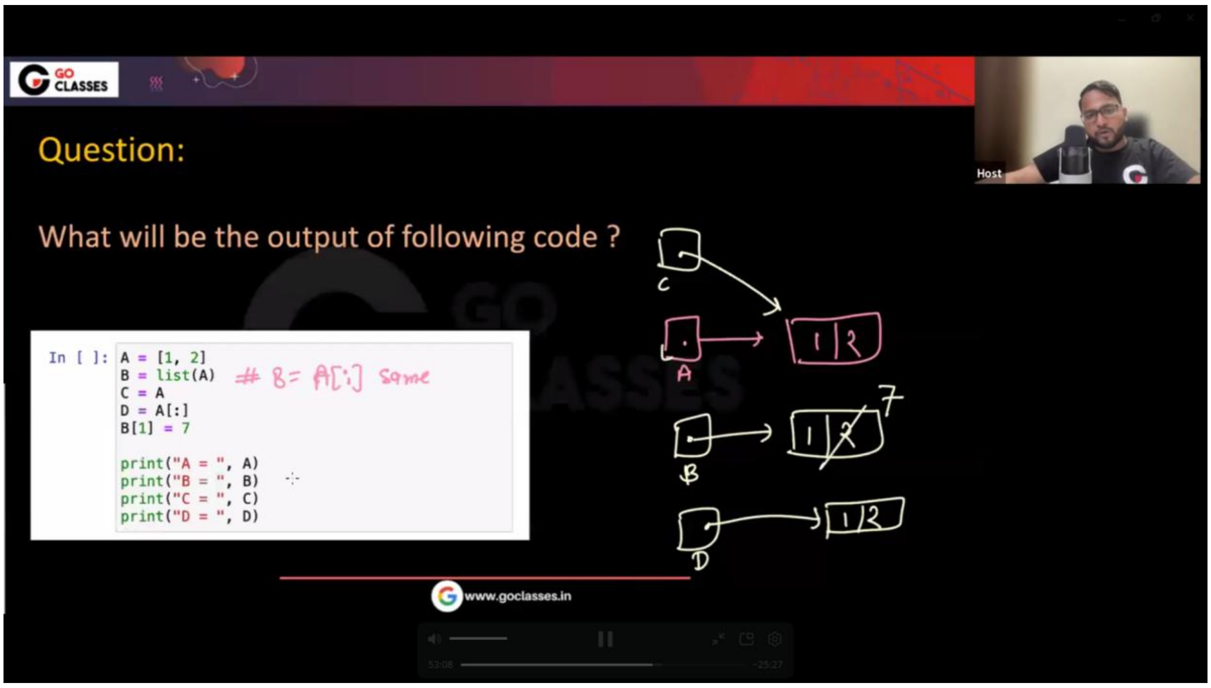
```
In [5]: L1 = [1,2,3]
        L2 = L1
        L3 = L1+[4]

        L1.append(5)

        print("L1 = ", L1)
        print("L2 = ", L2)
        print("L3 = ", L3)

        L1 =  [1, 2, 3, 5]
        L2 =  [1, 2, 3, 5]
        L3 =  [1, 2, 3, 4]

In [3]: L1 = [1,2,3]
        L2 = L1
```

Chrome File Edit View History Bookmarks Profiles Tab Window Help
Wed 17 Jul 9:14 PM
Course Builder
Python Programming GO Cla
localhost:8888/notebooks/Documents/GATE%20DA/Python%20Programming%20GO%20Classes.ipynb
Run
Code
Host
50:45
-27:50



```
In [5]: L1 = [1,2,3]
        L2 = L1
        L3 = L1+[4]

        L1.append(5)

        print("L1 = ", L1)
        print("L2 = ", L2)
        print("L3 = ", L3)

        L1 =  [1, 2, 3, 5]
        L2 =  [1, 2, 3, 5]
        L3 =  [1, 2, 3, 4]
```

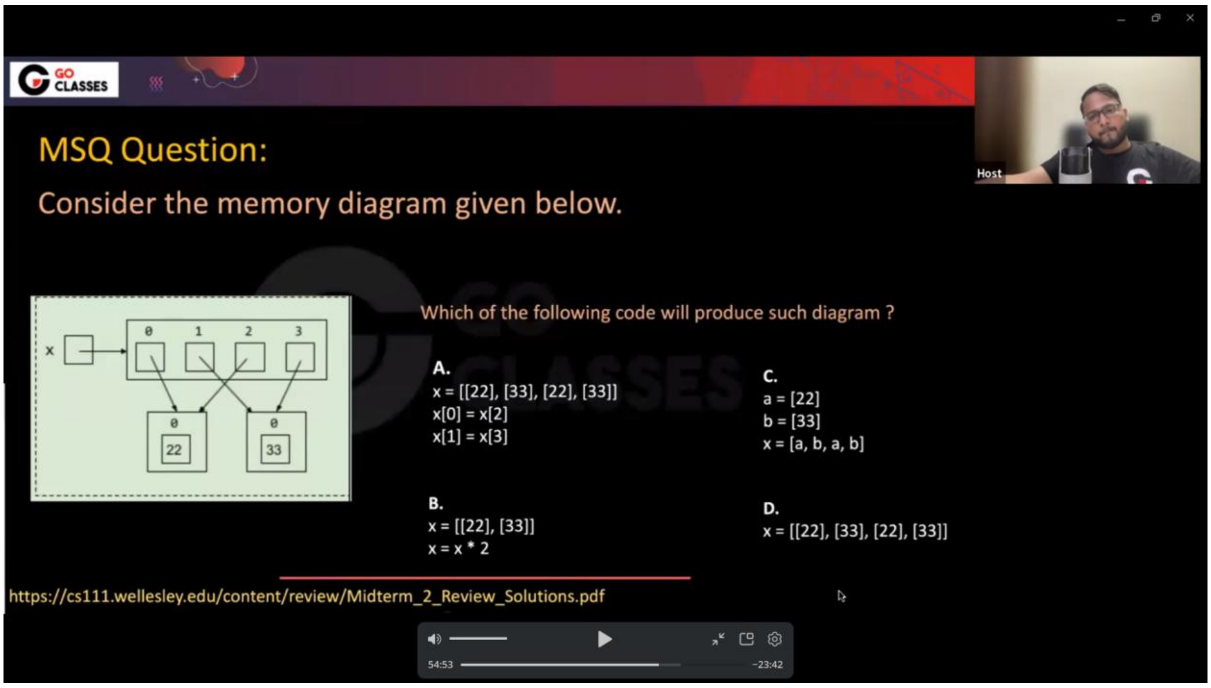
append will not create a new list, it will append inplace.



## Question:

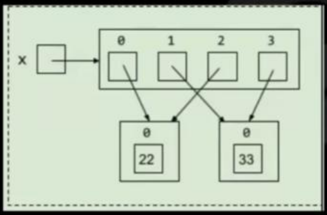
What will be the output of following code ?

```
In [ ]: A = [1, 2]
        B = list(A)     # B = A[:] same
        C = A
        D = A[:]
        B[1] = 7

        print("A = ", A)
        print("B = ", B)
        print("C = ", C)
        print("D = ", D)
```



## MSQ Question:

Consider the memory diagram given below.



Which of the following code will produce such diagram ?

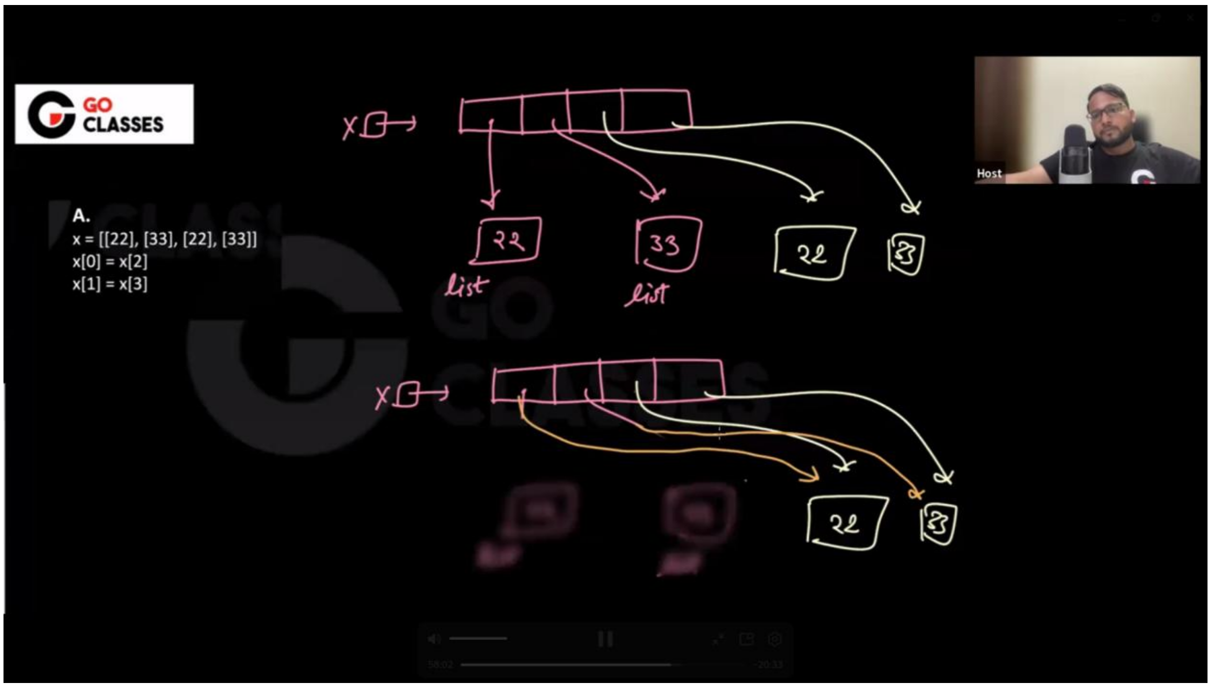
**A.**
```
x = [[22], [33], [22], [33]]
x[0] = x[2]
x[1] = x[3]
```

**B.**
```
x = [[22], [33]]
x = x * 2
```

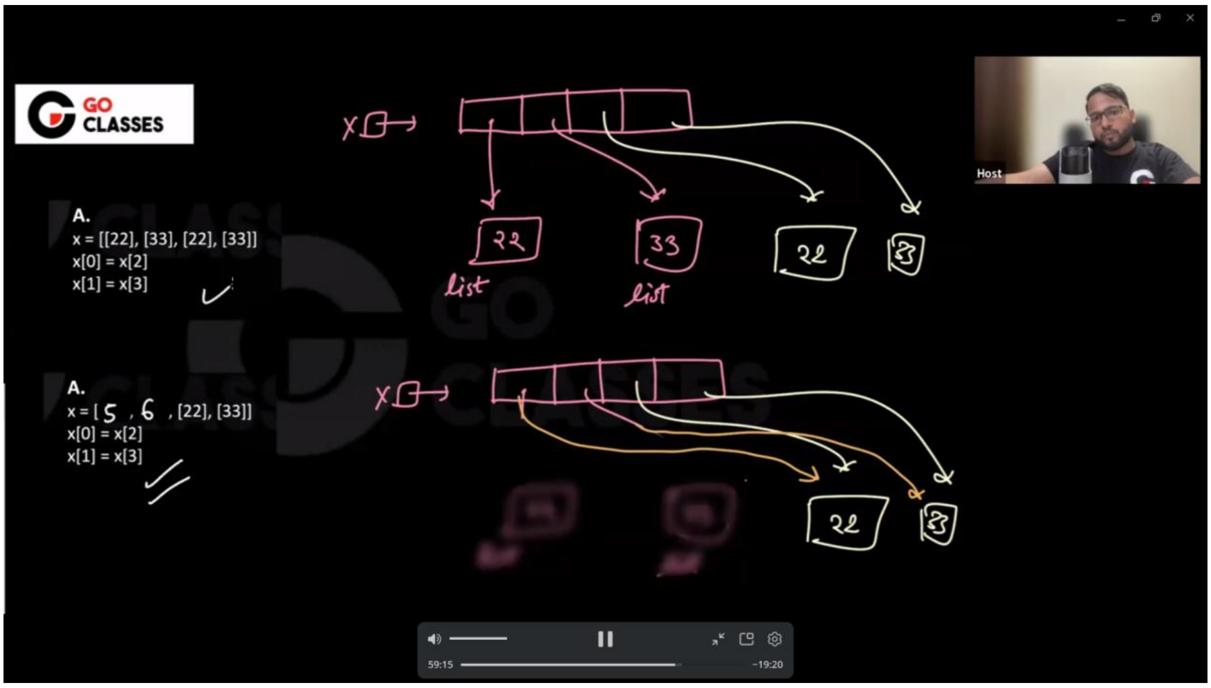
**C.**
```
a = [22]
b = [33]
x = [a, b, a, b]
```

**D.**
```
x = [[22], [33], [22], [33]]
```

https://cs111.wellesley.edu/content/review/Midterm_2_Review_Solutions.pdf

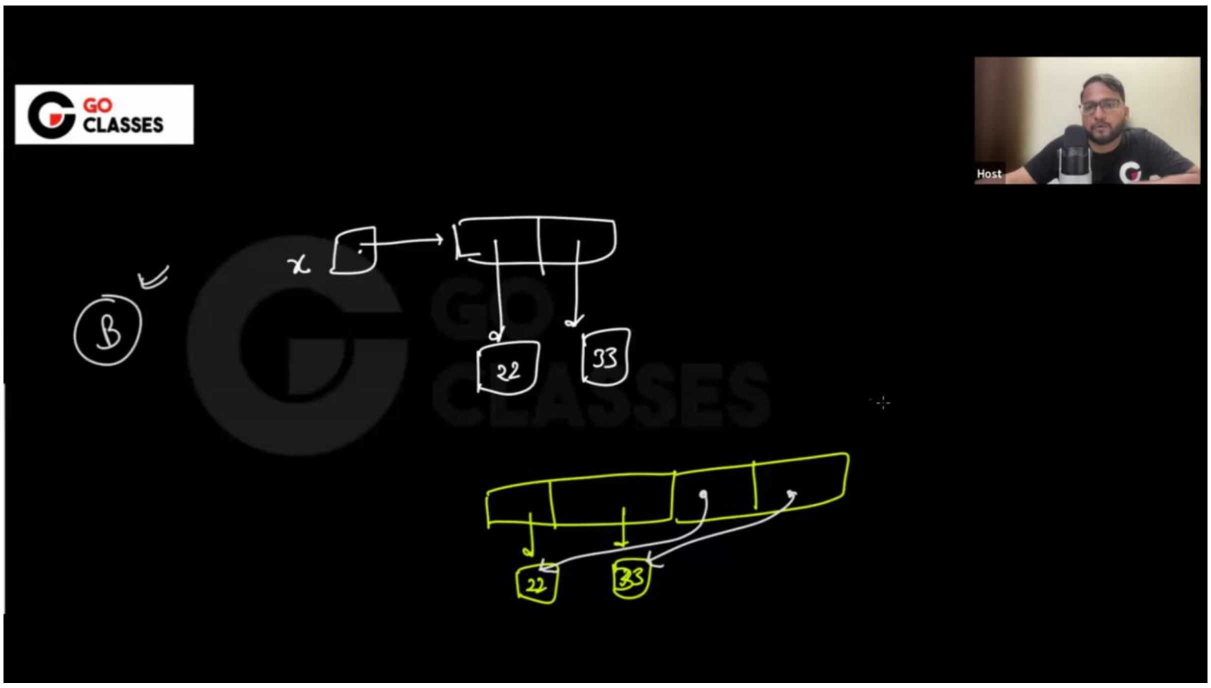


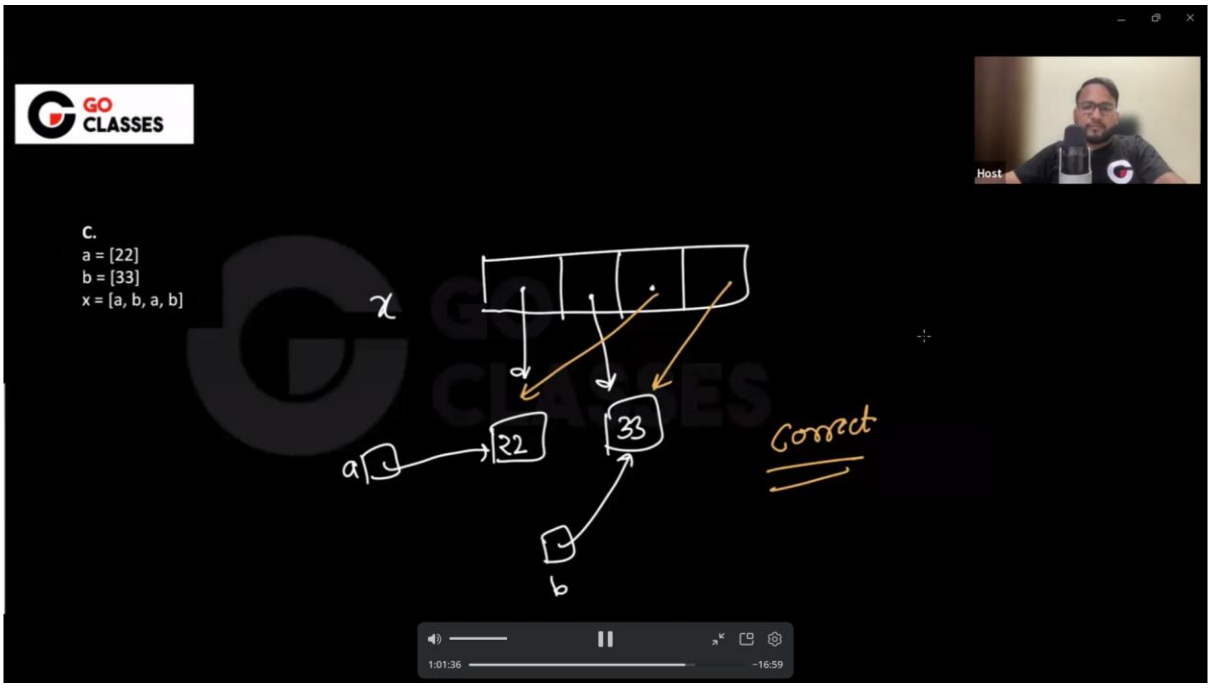
**A.**
```
x = [[22], [33], [22], [33]]
x[0] = x[2]
x[1] = x[3]
```



**A.**
x = [[22], [33], [22], [33]]
x[0] = x[2]
x[1] = x[3]
✓

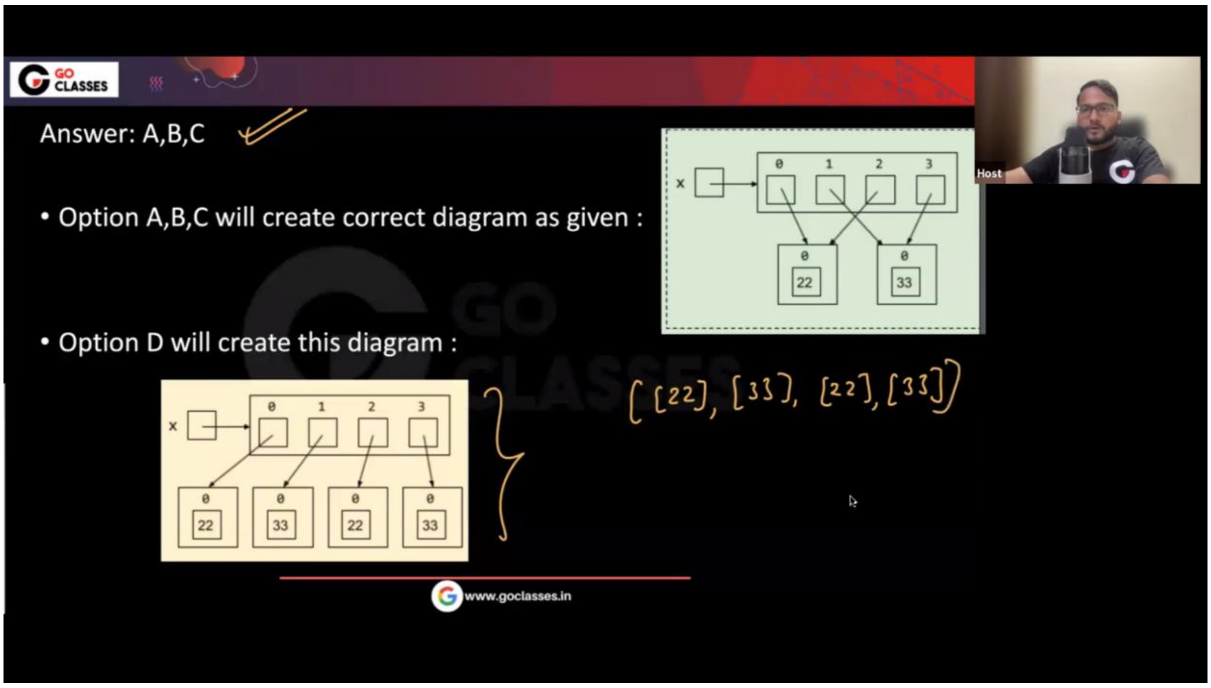
**A.**
x = [ 5 , 6 , [22], [33]]
x[0] = x[2]
x[1] = x[3]
✓✓





C.
```
a = [22]
b = [33]
x = [a, b, a, b]
```

Correct

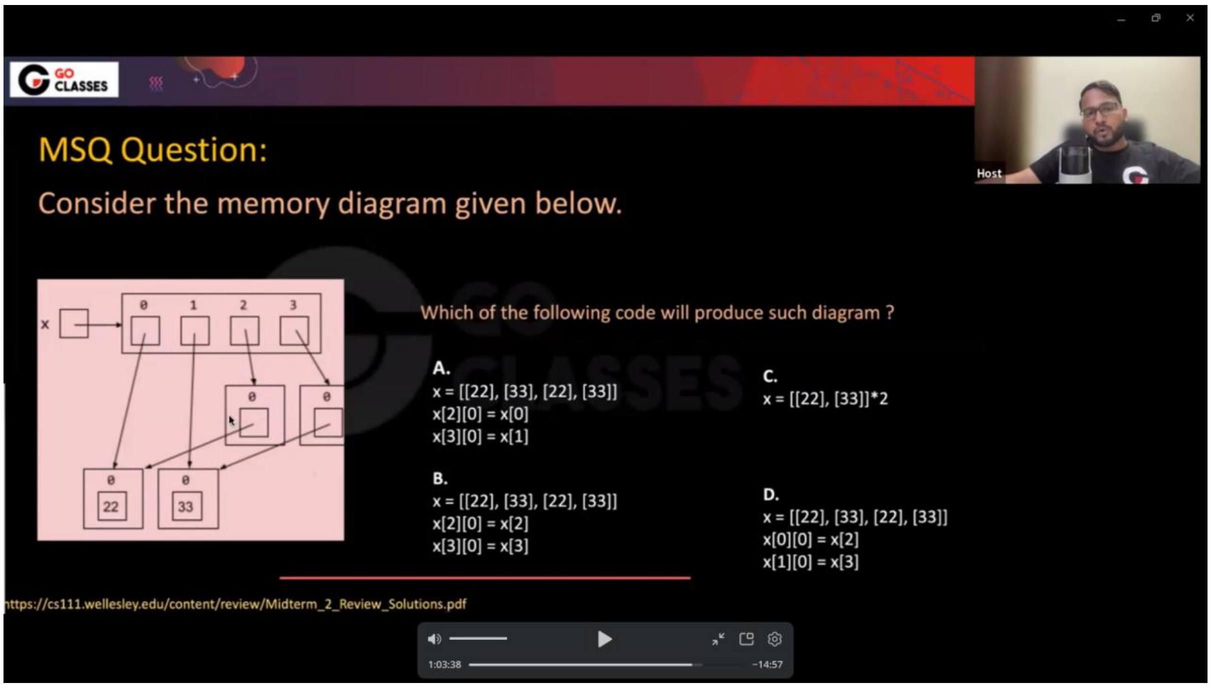


Answer: A,B,C ✓

- Option A,B,C will create correct diagram as given :
- Option D will create this diagram :

[[22], [33], [22], [33]]

www.goclasses.in



## MSQ Question:

Consider the memory diagram given below.

Which of the following code will produce such diagram ?

**A.**
```
x = [[22], [33], [22], [33]]
x[2][0] = x[0]
x[3][0] = x[1]
```

**B.**
```
x = [[22], [33], [22], [33]]
x[2][0] = x[2]
x[3][0] = x[3]
```

**C.**
```
x = [[22], [33]]*2
```

**D.**
```
x = [[22], [33], [22], [33]]
x[0][0] = x[2]
x[1][0] = x[3]
```

https://cs111.wellesley.edu/content/review/Midterm_2_Review_Solutions.pdf

1:03:38 -14:57

### Option A

python Copy code

```python
x = [[22], [33], [22], [33]]
x[2][0] = x[0]
x[3][0] = x[1]
```

After these assignments:

- `x[2][0]` becomes `[22]` (a list inside a list — different structure).
  ❌ So this **doesn't match** the diagram (we'd have nested lists).

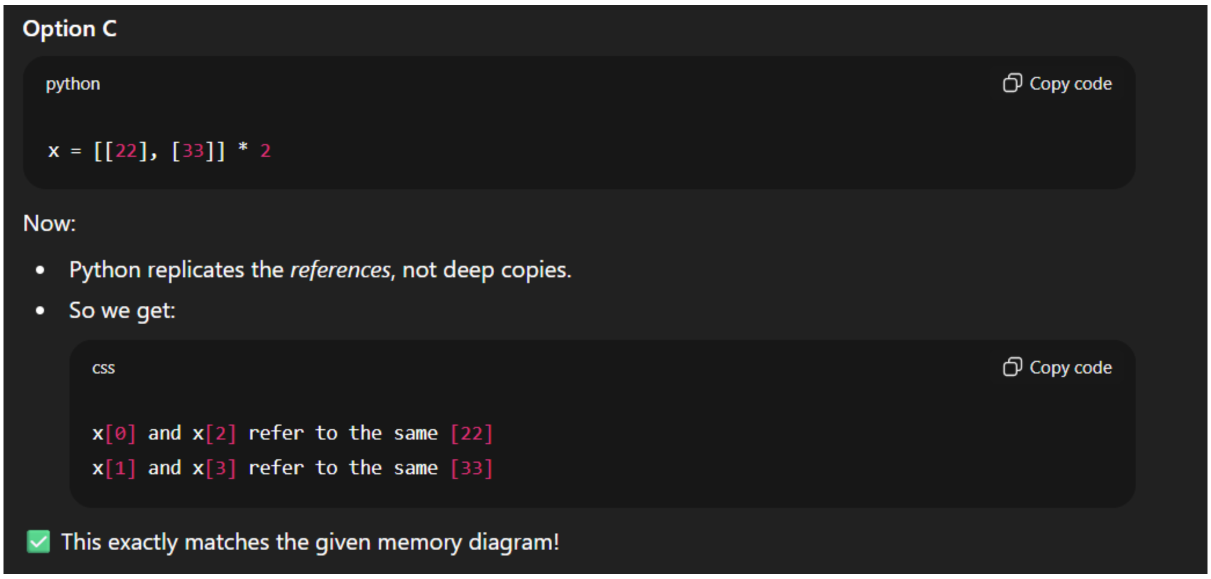
### Option B

python Copy code

```python
x = [[22], [33], [22], [33]]
x[2][0] = x[2]
x[3][0] = x[3]
```

Here each inner list refers to itself → **self-reference**, not sharing.

❌ Doesn't match.



**Option C**

python

```python
x = [[22], [33]] * 2
```

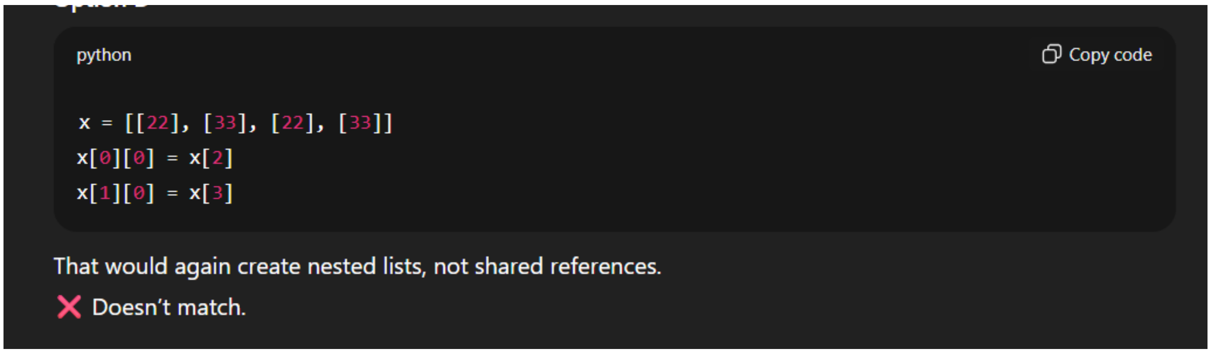
Now:

- Python replicates the *references*, not deep copies.
- So we get:

  css

  ```css
  x[0] and x[2] refer to the same [22]
  x[1] and x[3] refer to the same [33]
  ```

✅ This exactly matches the given memory diagram!

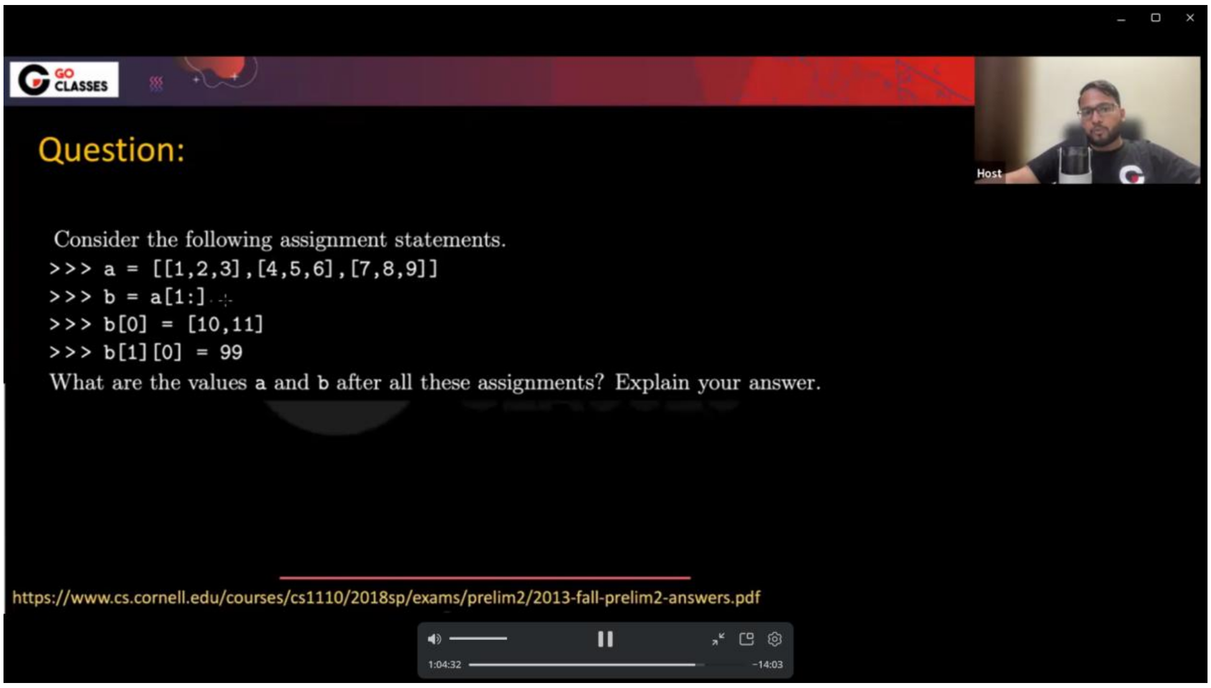


python

```python
x = [[22], [33], [22], [33]]
x[0][0] = x[2]
x[1][0] = x[3]
```

That would again create nested lists, not shared references.

❌ Doesn't match.



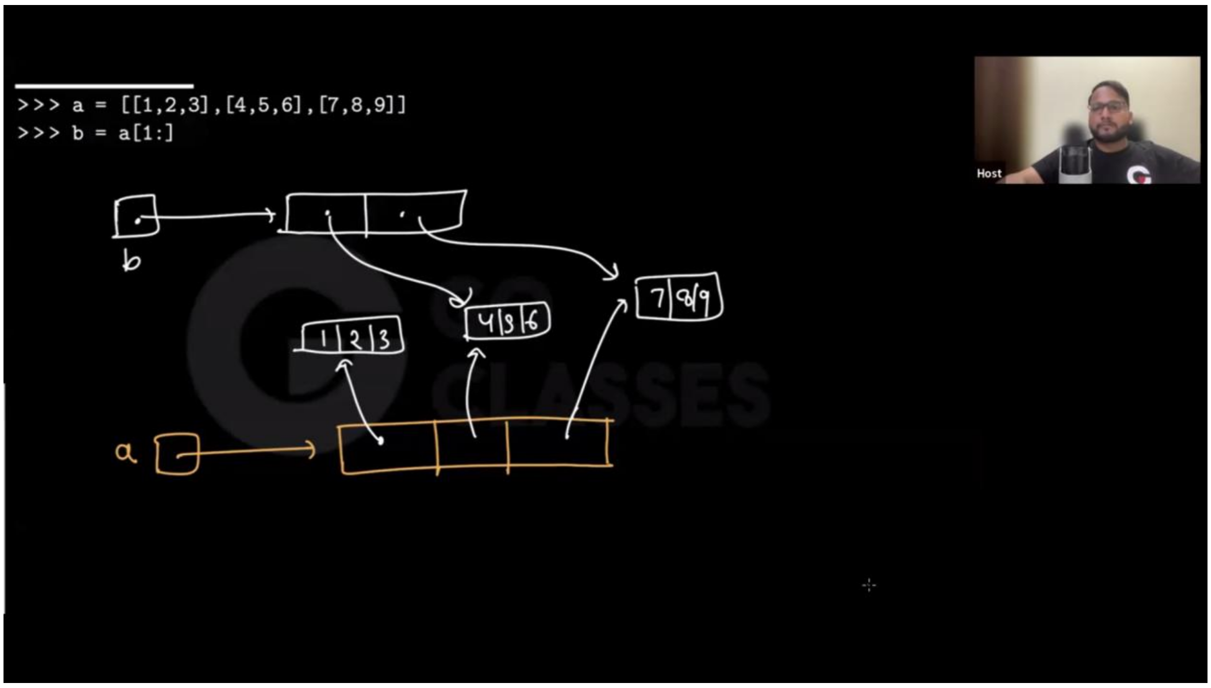
## Question:

Consider the following assignment statements.

```
>>> a = [[1,2,3],[4,5,6],[7,8,9]]
>>> b = a[1:]
>>> b[0] = [10,11]
>>> b[1][0] = 99
```

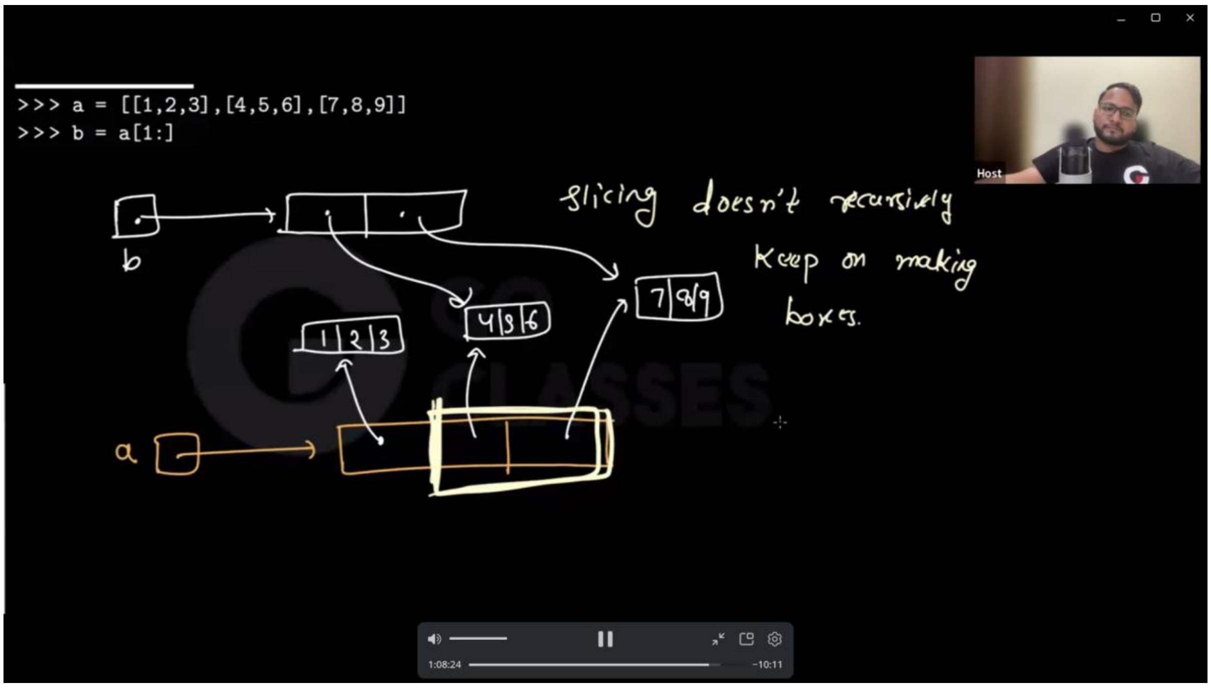
What are the values a and b after all these assignments? Explain your answer.

https://www.cs.cornell.edu/courses/cs1110/2018sp/exams/prelim2/2013-fall-prelim2-answers.pdf



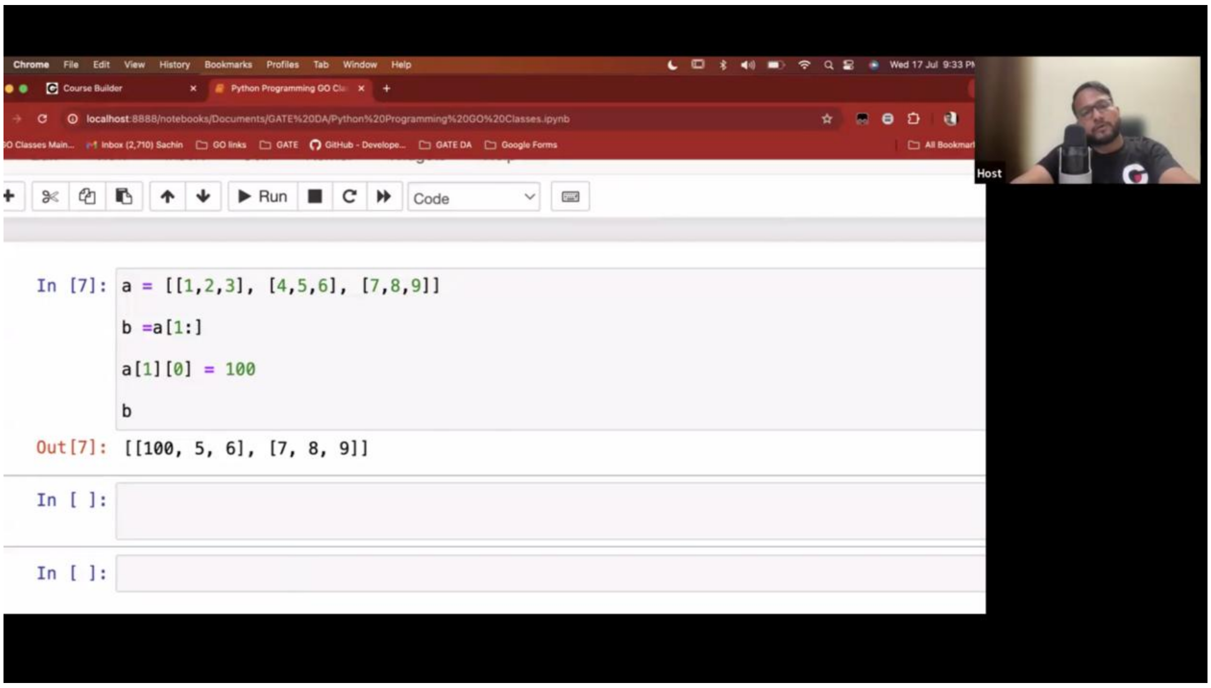
```
>>> a = [[1,2,3],[4,5,6],[7,8,9]]
>>> b = a[1:]
```

Slicing makes only one box(not recursively so on…)

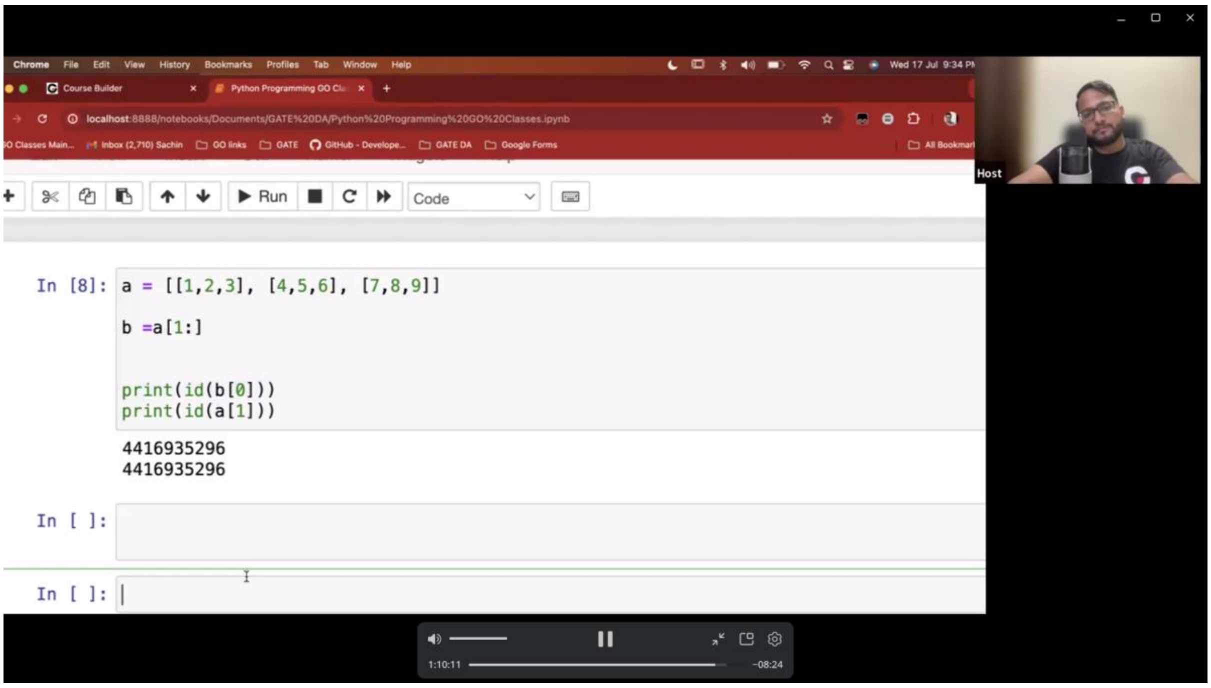


```
>>> a = [[1,2,3],[4,5,6],[7,8,9]]
>>> b = a[1:]
```

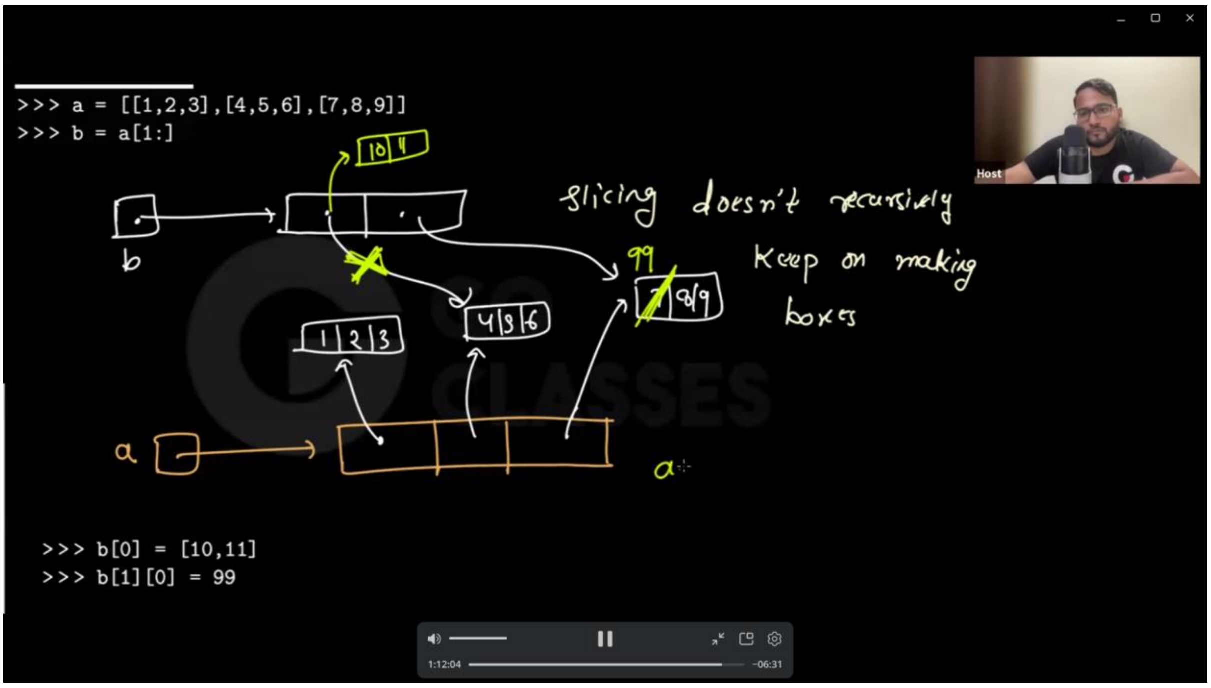
slicing doesn't recursively Keep on making boxes.



```
In [7]: a = [[1,2,3], [4,5,6], [7,8,9]]

        b =a[1:]

        a[1][0] = 100

        b

Out[7]: [[100, 5, 6], [7, 8, 9]]

In [ ]:

In [ ]:
```



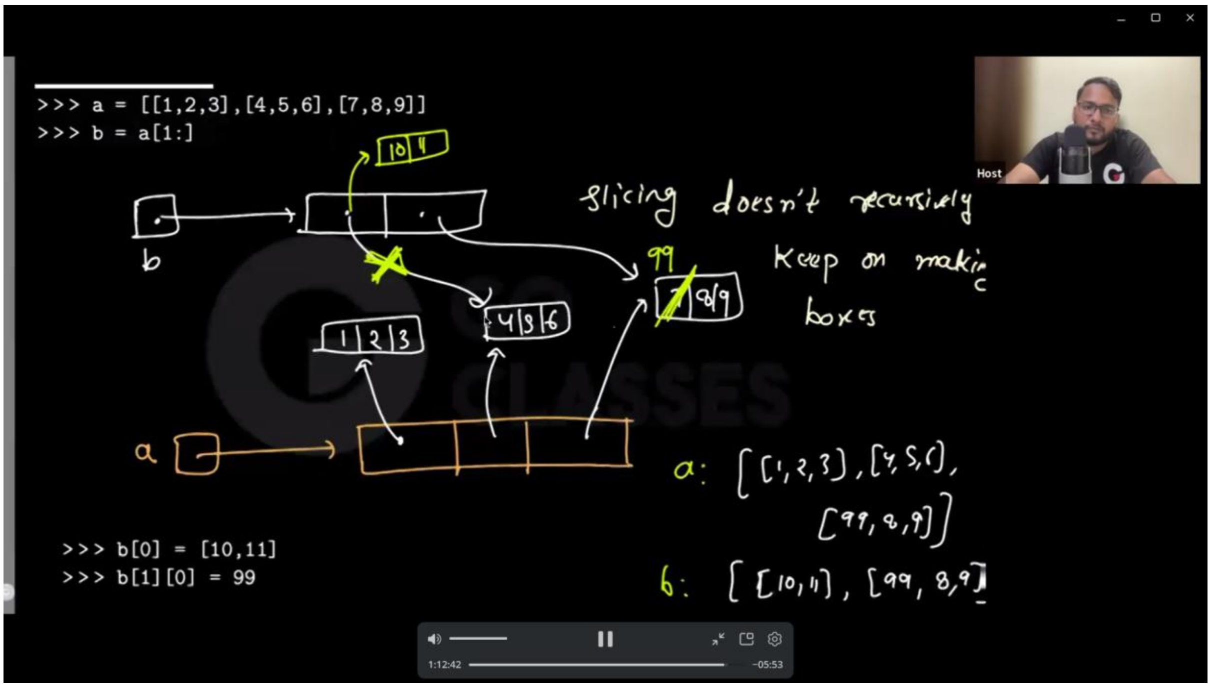
```
In [8]: a = [[1,2,3], [4,5,6], [7,8,9]]

        b =a[1:]


        print(id(b[0]))
        print(id(a[1]))

        4416935296
        4416935296

In [ ]:

In [ ]:
```



```
>>> a = [[1,2,3],[4,5,6],[7,8,9]]
>>> b = a[1:]
```

slicing doesn't recursively keep on making boxes

```
>>> b[0] = [10,11]
>>> b[1][0] = 99
```



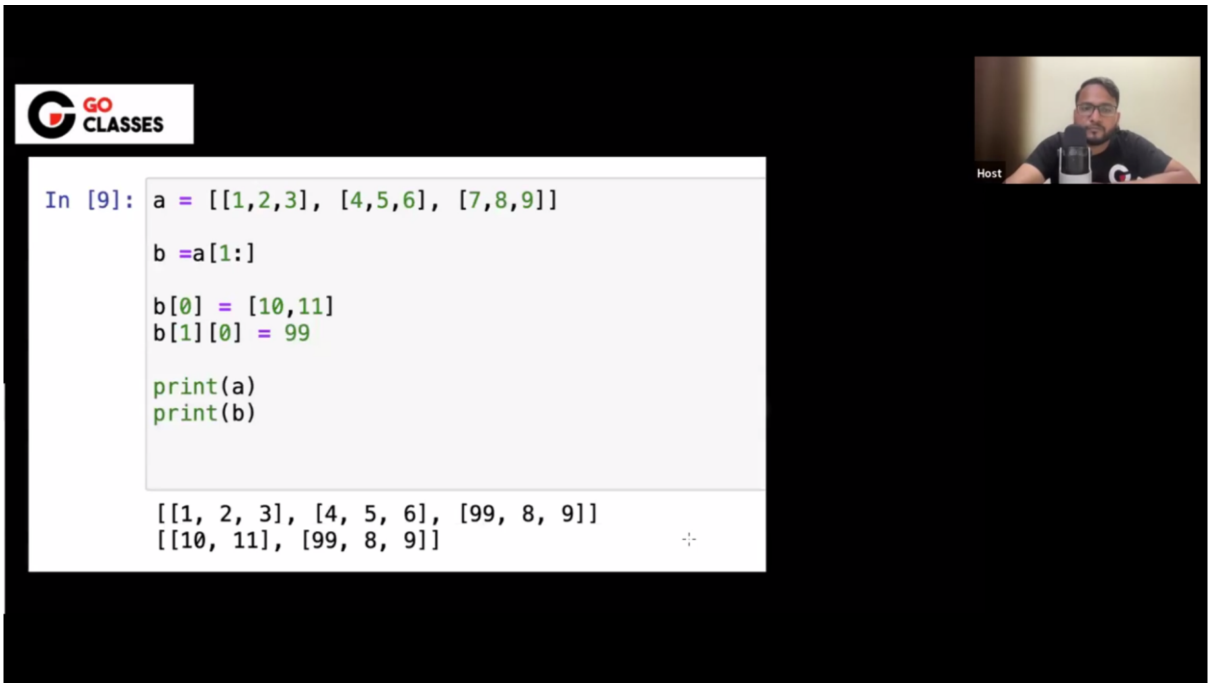
```
>>> a = [[1,2,3],[4,5,6],[7,8,9]]
>>> b = a[1:]
```

slicing doesn't recursively keep on making boxes

a: [[1,2,3],[4,5,6],
 [99,8,9]]

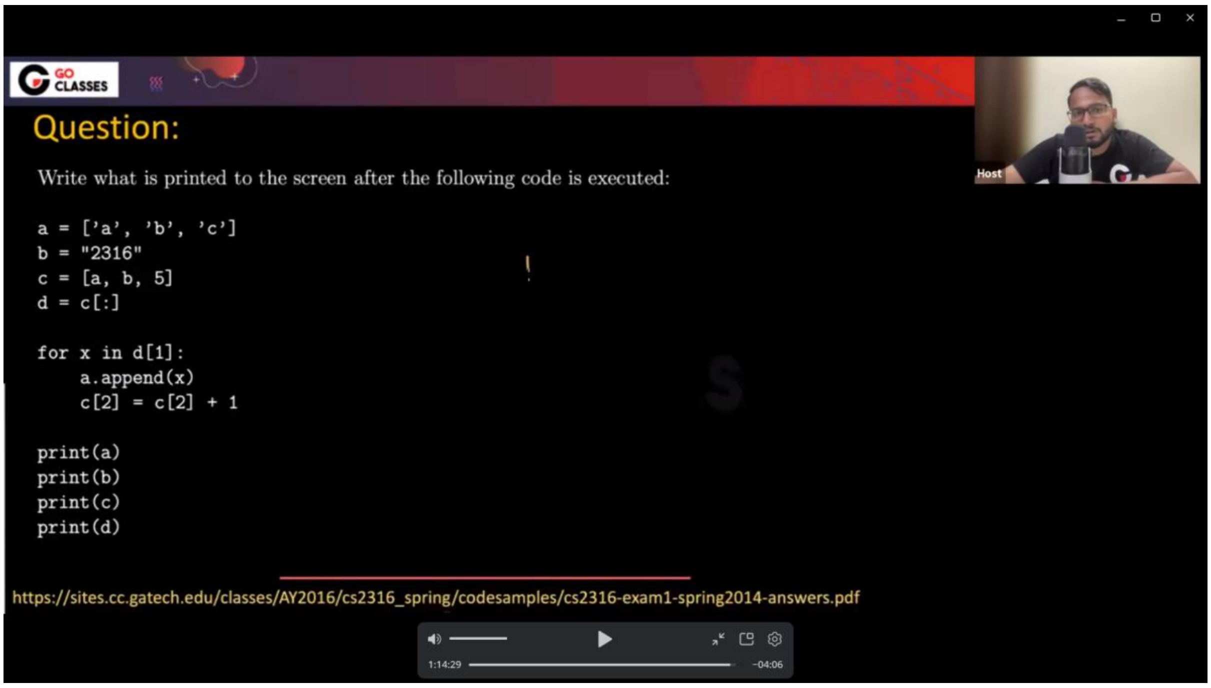
b: [[10,11], [99, 8,9]]

```
>>> b[0] = [10,11]
>>> b[1][0] = 99
```



```
In [9]: a = [[1,2,3], [4,5,6], [7,8,9]]

        b =a[1:]

        b[0] = [10,11]
        b[1][0] = 99

        print(a)
        print(b)
```

```
[[1, 2, 3], [4, 5, 6], [99, 8, 9]]
[[10, 11], [99, 8, 9]]
```

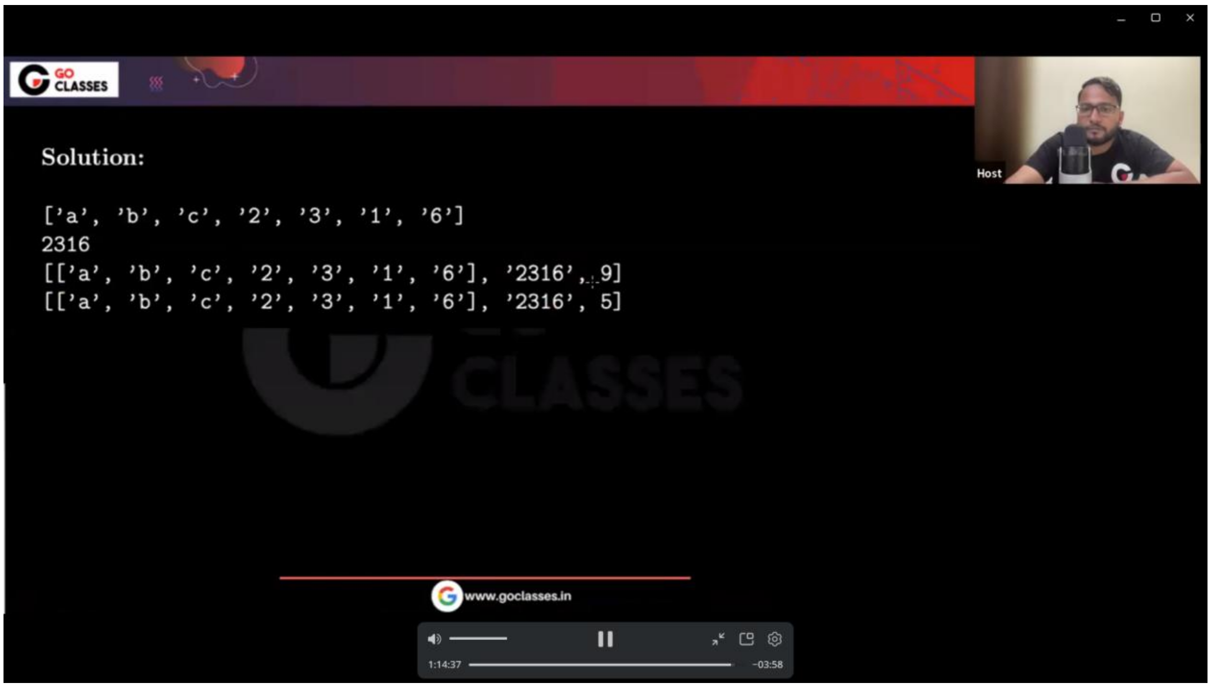


## Question:

Write what is printed to the screen after the following code is executed:

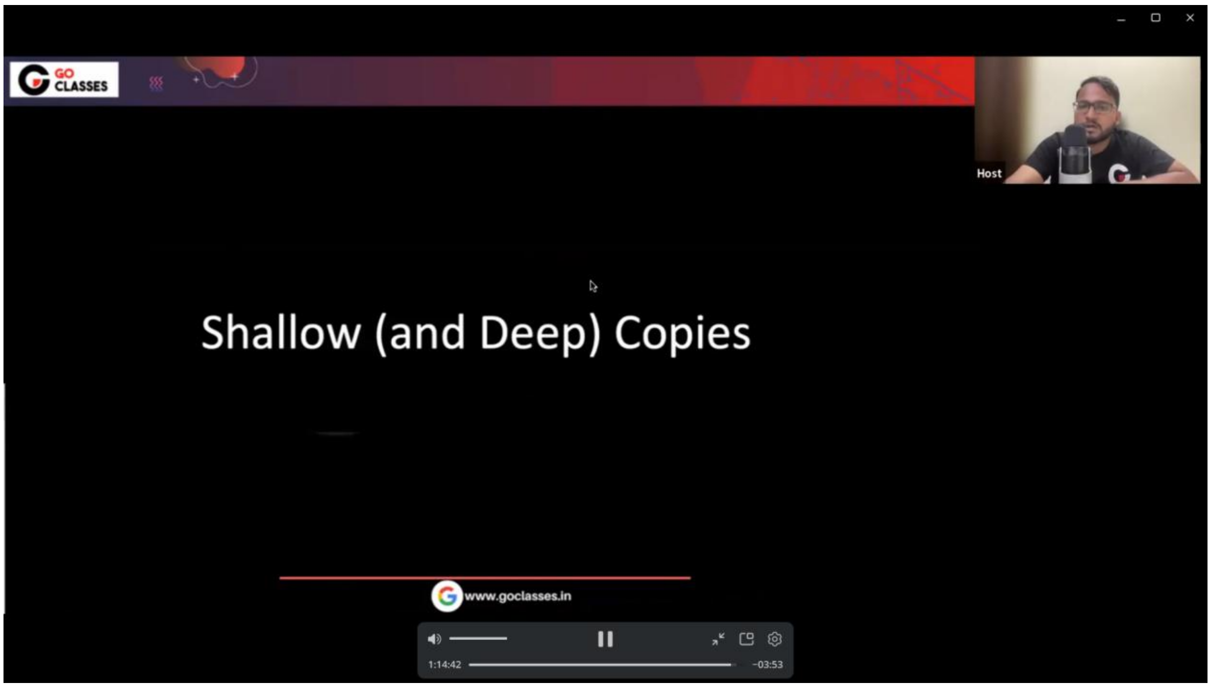
```
a = ['a', 'b', 'c']
b = "2316"
c = [a, b, 5]
d = c[:]

for x in d[1]:
    a.append(x)
    c[2] = c[2] + 1

print(a)
print(b)
print(c)
print(d)
```

https://sites.cc.gatech.edu/classes/AY2016/cs2316_spring/codesamples/cs2316-exam1-spring2014-answers.pdf

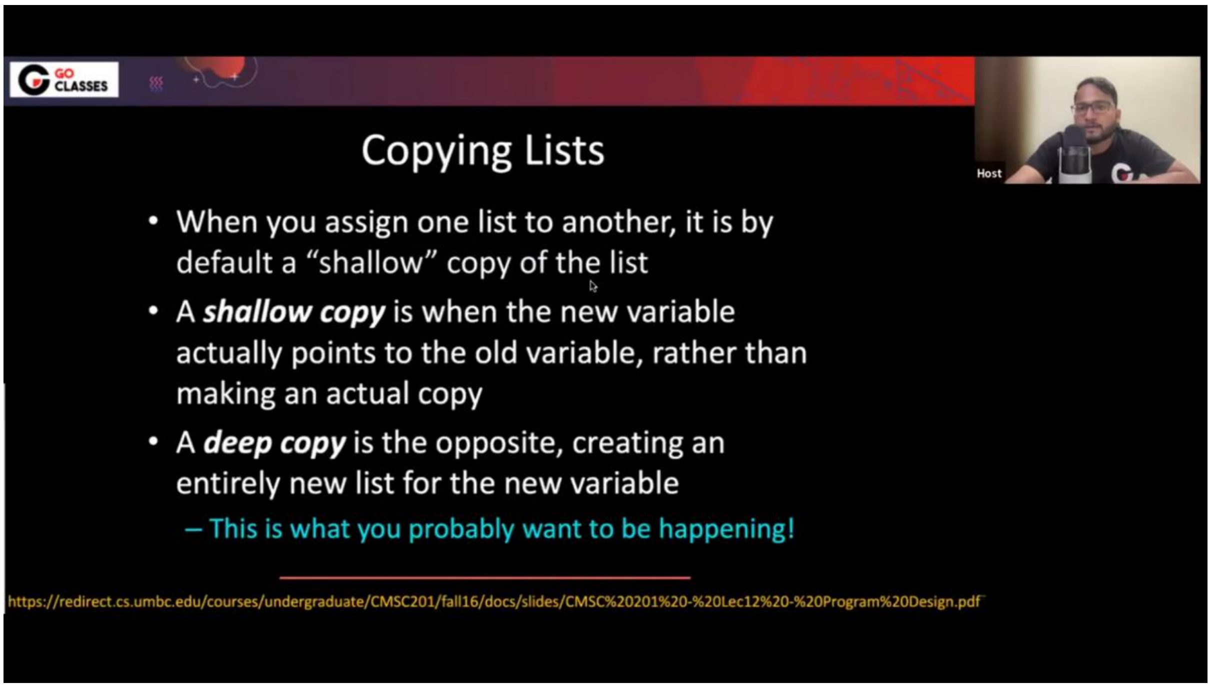


**Solution:**

```
['a', 'b', 'c', '2', '3', '1', '6']
2316
[['a', 'b', 'c', '2', '3', '1', '6'], '2316', 9]
[['a', 'b', 'c', '2', '3', '1', '6'], '2316', 5]
```



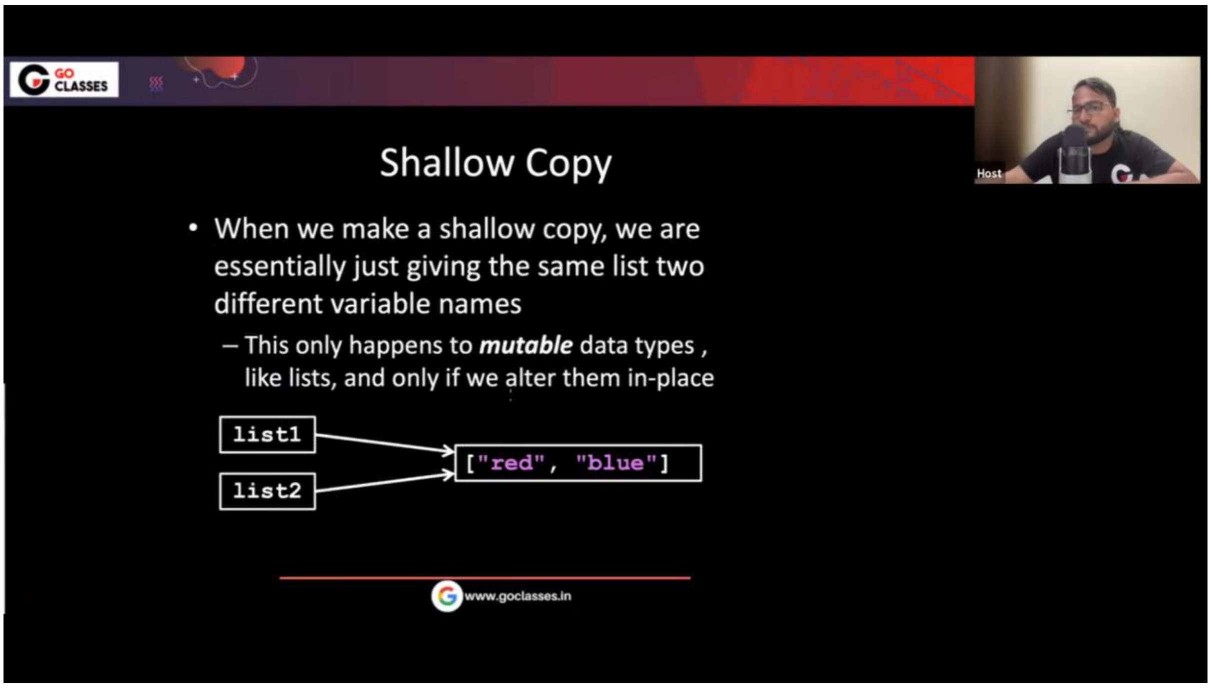
## Shallow (and Deep) Copies



## Copying Lists

- When you assign one list to another, it is by default a "shallow" copy of the list
- A ***shallow copy*** is when the new variable actually points to the old variable, rather than making an actual copy
- A ***deep copy*** is the opposite, creating an entirely new list for the new variable
  - This is what you probably want to be happening!

https://redirect.cs.umbc.edu/courses/undergraduate/CMSC201/fall16/docs/slides/CMSC%20201%20-%20Lec12%20-%20Program%20Design.pdf

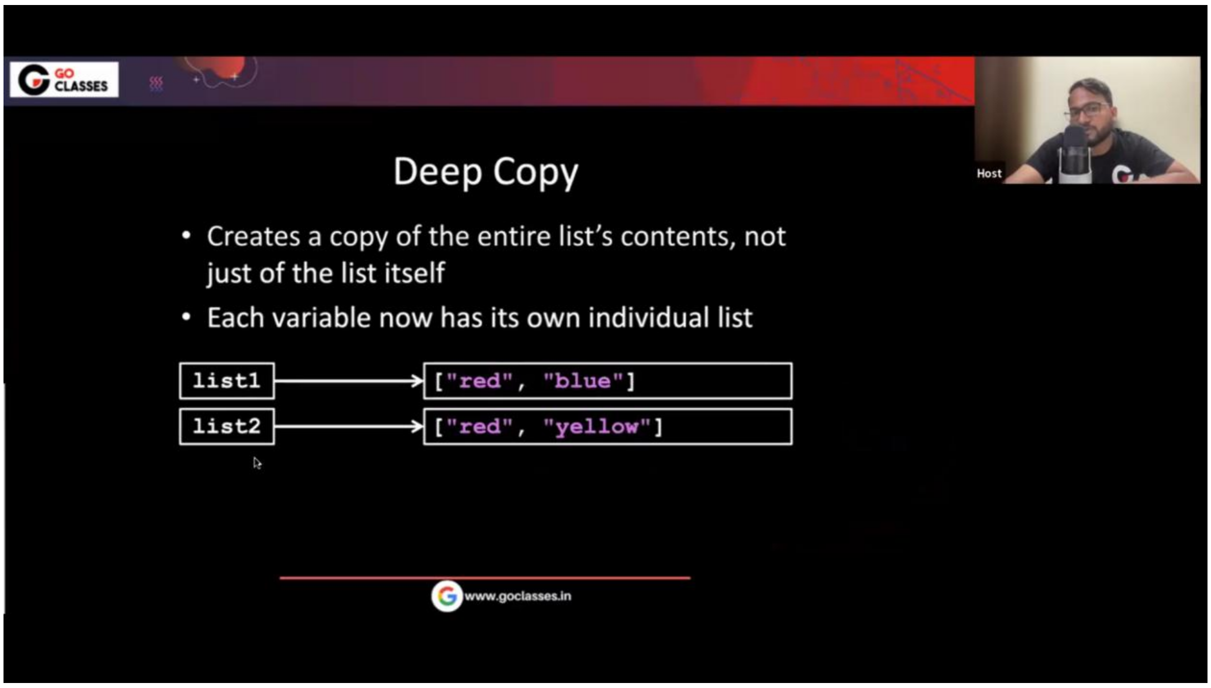


## Shallow Copy

- When we make a shallow copy, we are essentially just giving the same list two different variable names
  - This only happens to ***mutable*** data types , like lists, and only if we alter them in-place

$$\begin{tikzcd} \boxed{\texttt{list1}} \arrow[dr] & \\ & \boxed{\texttt{["red", "blue"]}} \\ \boxed{\texttt{list2}} \arrow[ur] & \end{tikzcd}$$

www.goclasses.in

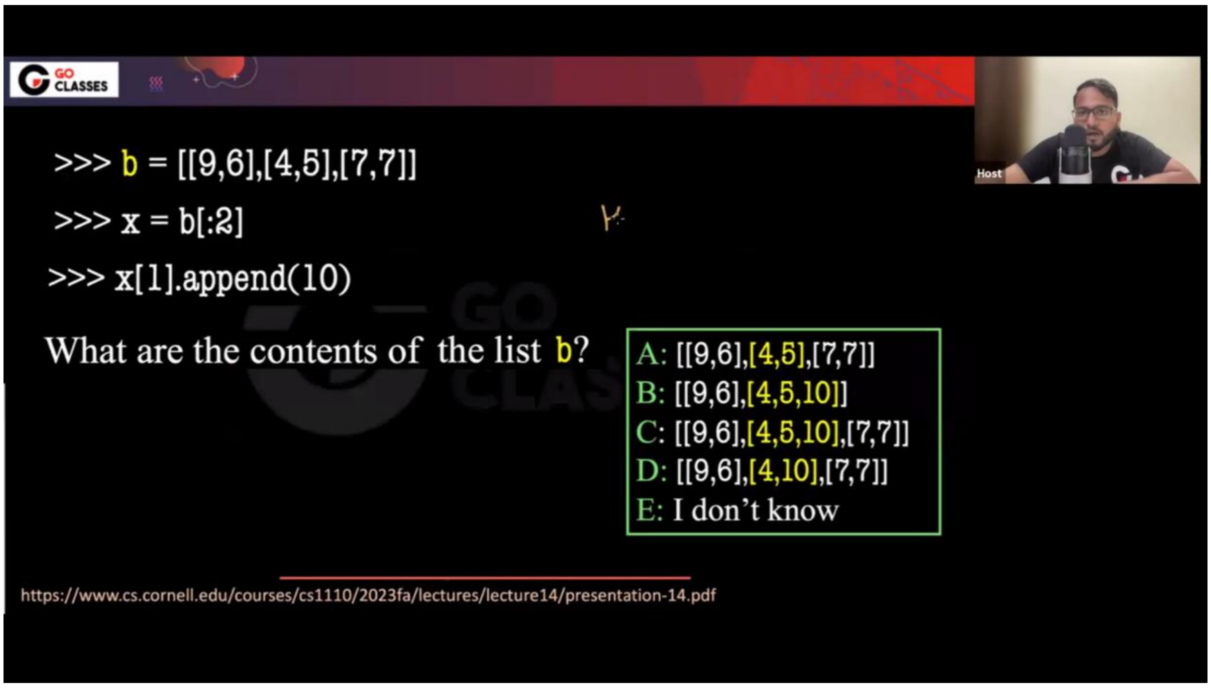


## Deep Copy

- Creates a copy of the entire list's contents, not just of the list itself
- Each variable now has its own individual list

```
list1 → ["red", "blue"]
list2 → ["red", "yellow"]
```

www.goclasses.in

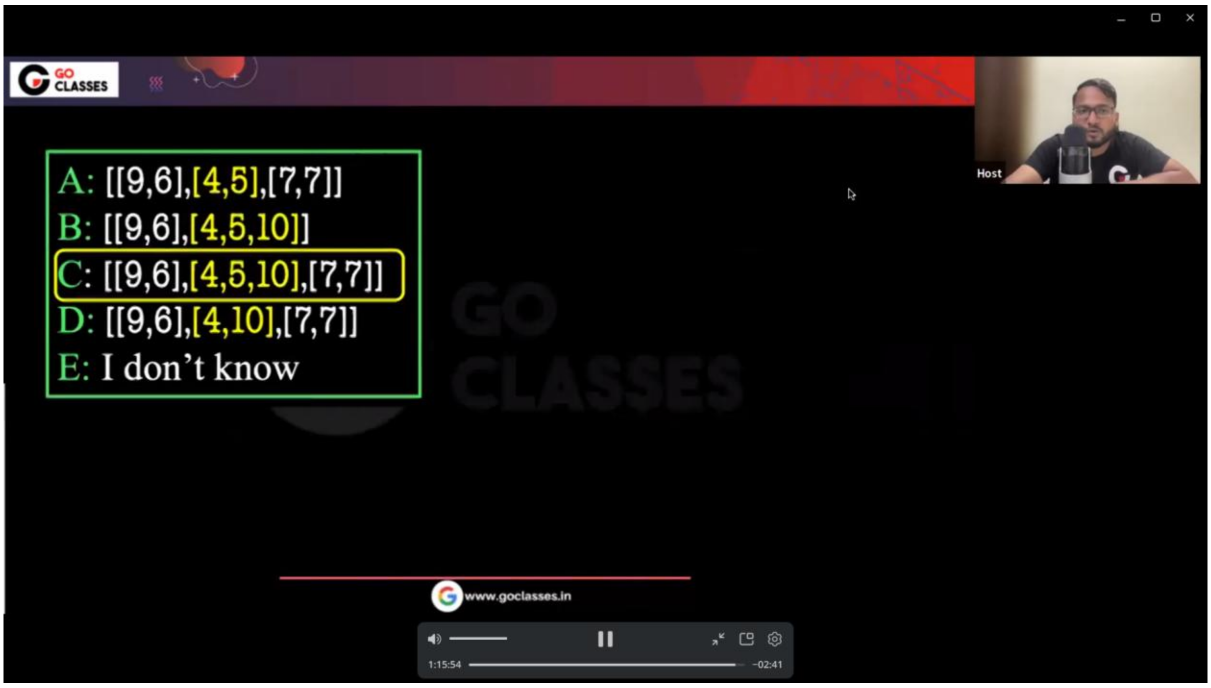


```
>>> b = [[9,6],[4,5],[7,7]]
>>> x = b[:2]
>>> x[1].append(10)
```

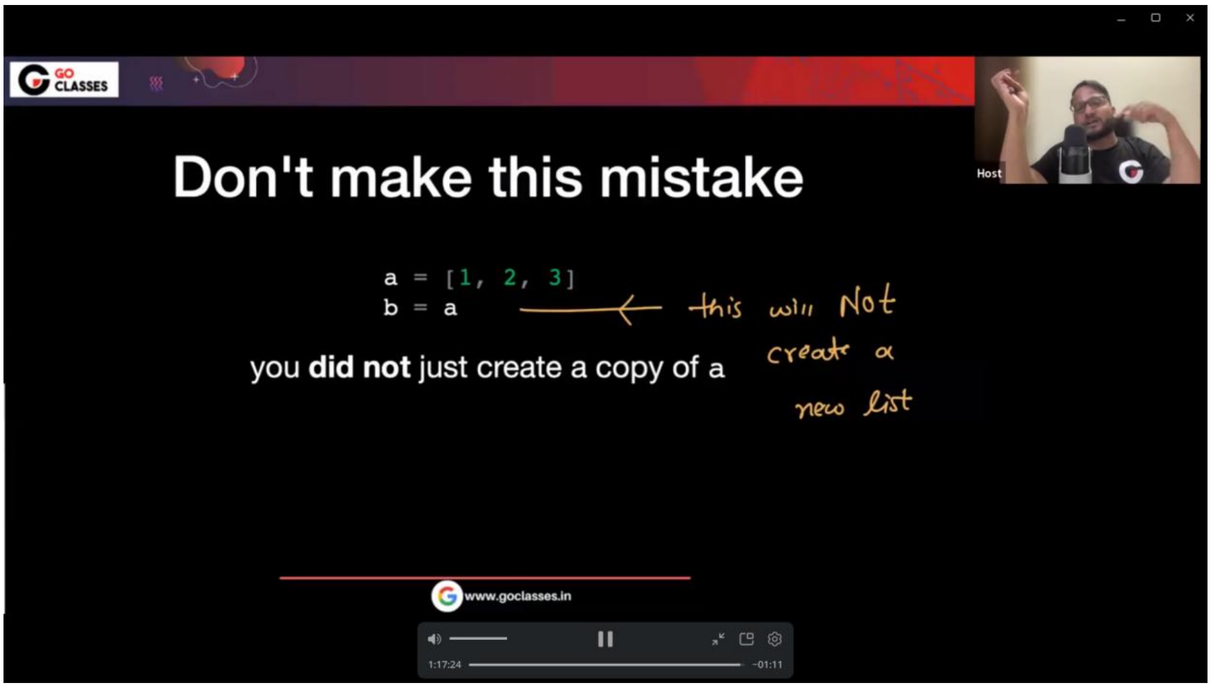
What are the contents of the list b?

A: [[9,6],[4,5],[7,7]]
B: [[9,6],[4,5,10]]
C: [[9,6],[4,5,10],[7,7]]
D: [[9,6],[4,10],[7,7]]
E: I don't know

https://www.cs.cornell.edu/courses/cs1110/2023fa/lectures/lecture14/presentation-14.pdf



A: [[9,6],[4,5],[7,7]]
B: [[9,6],[4,5,10]]
[✓ C: [[9,6],[4,5,10],[7,7]]]
D: [[9,6],[4,10],[7,7]]
E: I don't know



## Don't make this mistake

```
a = [1, 2, 3]
b = a
```

this will Not create a new list

you **did not** just create a copy of a