

When we talked about variables...

Sometimes I got lazy and wrote:

In [1]:  
number = 2

but what's truly happening is:

number → int → 2

everything in python is object.

www.goclasses.in

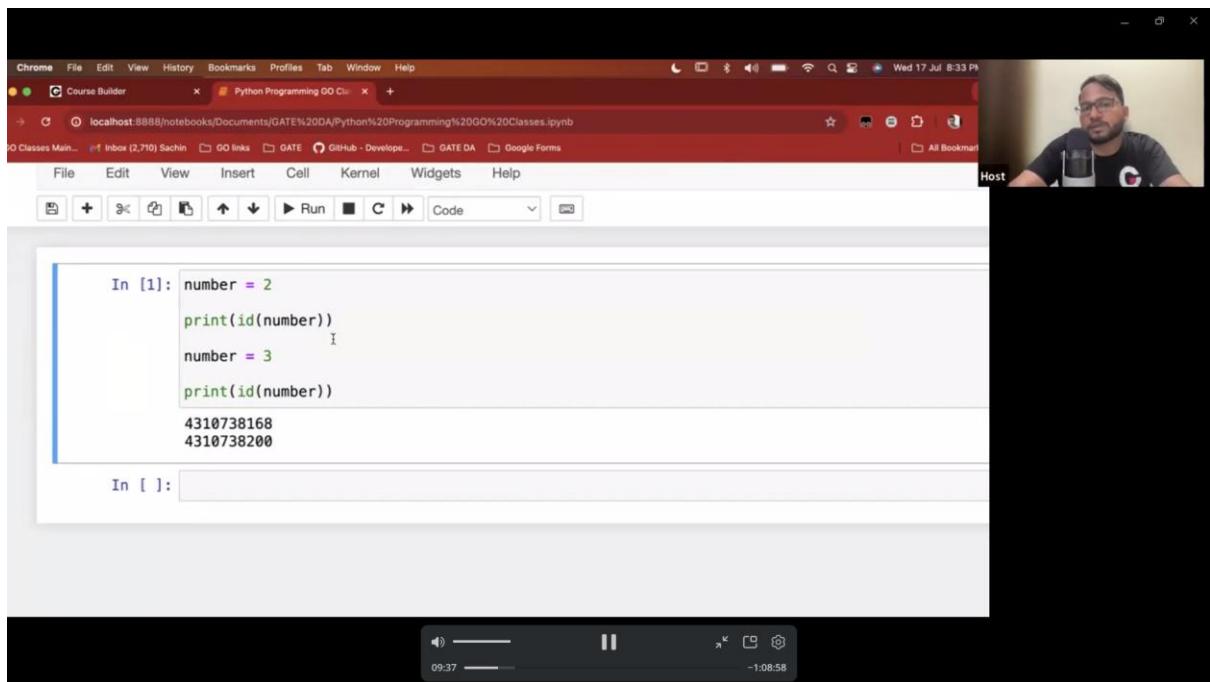
05:44 - 1:12:51

number = 2  
number = number + 1

The diagram shows a variable 'number' pointing to an integer object '2'. When 'number' is modified to 'number + 1', it does not create a new integer object but instead changes the value of the existing object '2' to '3'.

www.goclasses.in

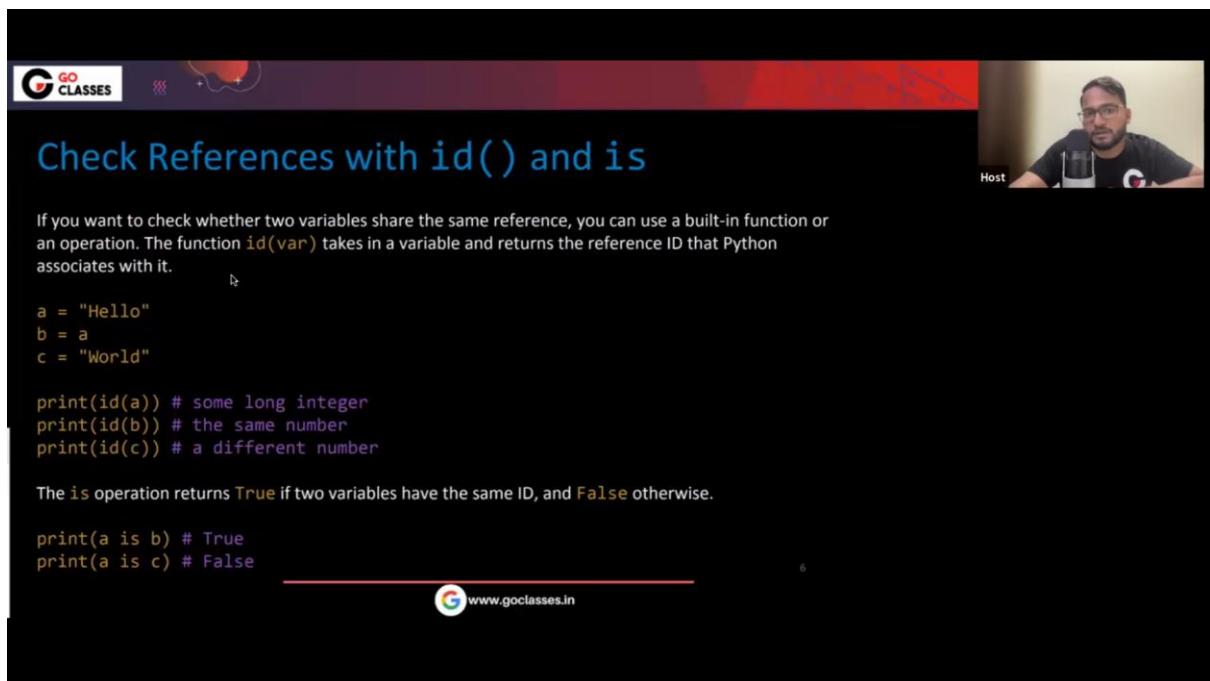
08:59 - 1:09:36



```
In [1]: number = 2
        print(id(number))
        I
        number = 3
        print(id(number))
4310738168
4310738200
```

In [ ]:

09:37 -1:08:58



## 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
```

---

www.goclasses.in

When we talked about variables...

Sometimes I got lazy and wrote:

In [1]:  
number = 2

but what's truly happening is:

number → int 2

All variables store **references** to **objects**.

Objects can have any type

www.goclasses.in

All variables store **references** to **objects**

In memory:

In code:

```
number = 2
number = 4
```

Like strings, ints 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
```

Host

www.goclasses.in

13:38 - 1:04:57

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

```
a = [4, 5]
b = a
b[0] = 1
print(a)
```

In [4]:

```
a = [4,5]
b = a
b[0] = 1
print(a)
[1, 5]
```

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

Host

In code:

```
a = [4, 5]
b = a
b[0] = 1
print(a)
[1, 5] # !!!
```

**In memory:**

The diagram illustrates the state of memory. A variable **a** is shown pointing to a list structure. This list has two elements at indices 0 and 1, containing the values 1 and 5 respectively. Another variable **b** is also shown pointing to the same list, indicating that both variables share the same memory location.

---

www.goclasses.in

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

The diagram shows the state of memory for the **weather** list after modification. The original list [63, "light rain", 8, "SSW", 29.75] is shown with each element having its type (int, str, int, str, float) and value. After the assignment **weather[1] = "cloudy"**, the list is updated to [63, "cloudy", 8, "SSW", 29.75]. The variable **weather** now points to this modified list. The original string "light rain" is still present in memory, but it is no longer referenced by the list.

---

www.goclasses.in

Host

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

www.goclasses.in

23:47 - 54:48

Question:

Make diagram having references to objects for the following code.

```
alist = ['a', 'b', 'c', 'd', 'e', 'f']
aslice = alist[2:4] ← Slicing makes new list.
print(aslice)
```

alist

aslice

www.goclasses.in

26:56 - 51:39

GO CLASSES

List slicing creates a new list

Host

```
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>

GO CLASSES

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)
```

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

www.goclasses.in

29:42 -48:53

GO CLASSES

Question:

What will be the output of following code ?

```
a = [4, 5]
b = a[:]
b[0] = 1
print(a)
```

slicing will create a different box for b.

www.goclasses.in

32:50 - 45:45

GO CLASSES

```
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]
```

Host

Question:

What will be the output of following code ?



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

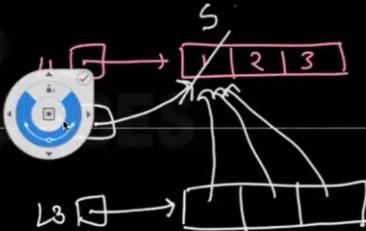
print(L3[0])
```

- 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]



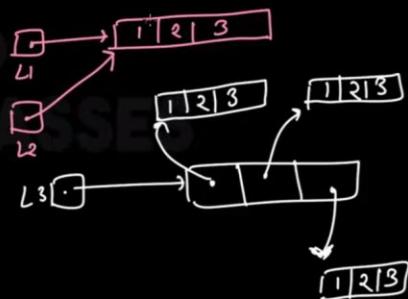
37:56 -> 40:39

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])
```



40:53 -> 43:42

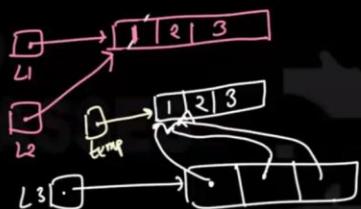
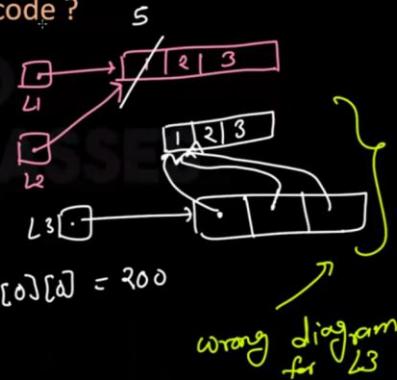


What will be the output of following code ?

```
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)
```

Host

www.goclasses.in

Host

```
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

50:45 -27:50

**GO CLASSES**

Host

```
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.

51:45 -26:50

**GO CLASSES**

Host

Question:

What will be the output of following code ?

```
In [1]: 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)
```

The diagram illustrates the state of variables after the code execution. It shows four boxes labeled C, A, B, and D. Box C contains '1'. Box A contains '1 | 2'. Box B contains '1 | 7'. Box D contains '1 | 2'. Arrows point from each label to its corresponding box. The box for B has a red 'X' over the number 2, indicating it is mutated.

www.goclasses.in

53:08 -25:27

**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](https://cs111.wellesley.edu/content/review/Midterm_2_Review_Solutions.pdf)

Host

**MSQ Question:**

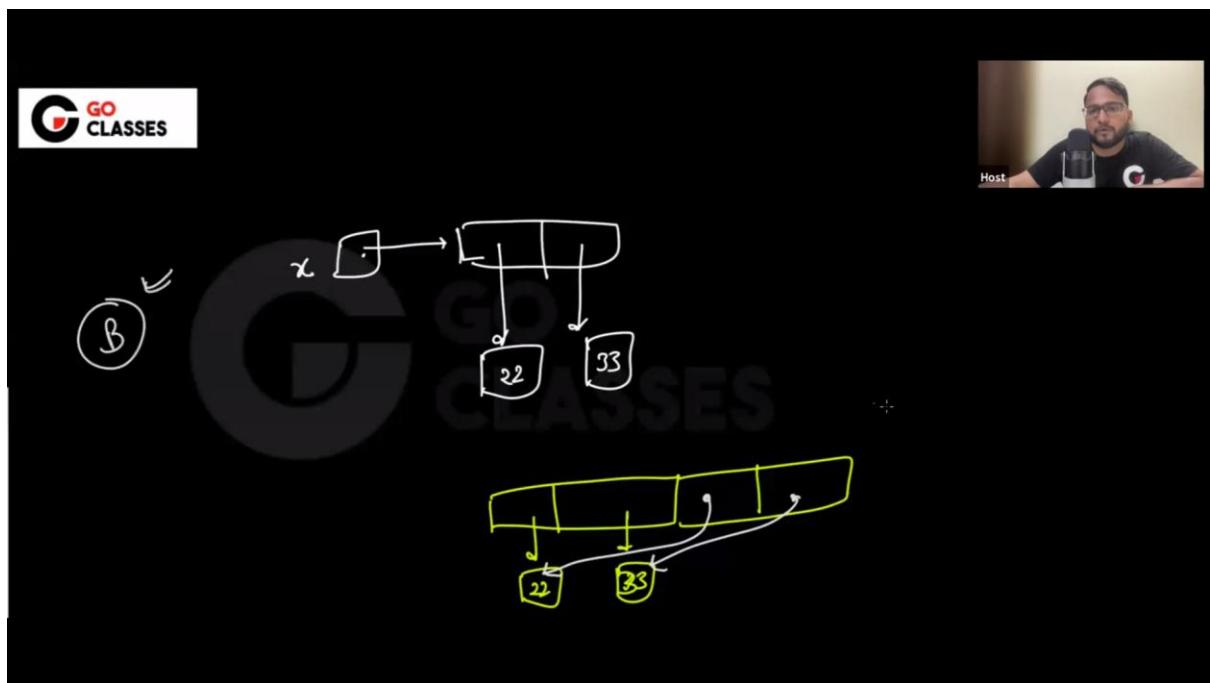
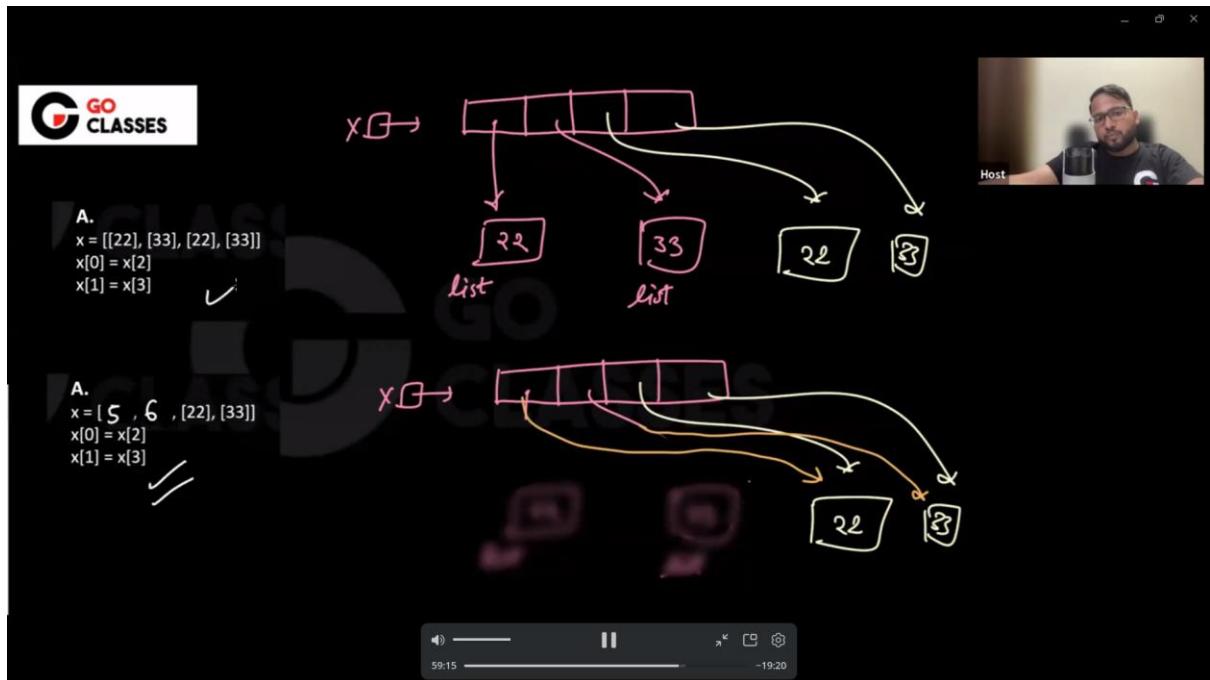
A.  
 $x = [[22], [33], [22], [33]]$   
 $x[0] = x[2]$   
 $x[1] = x[3]$

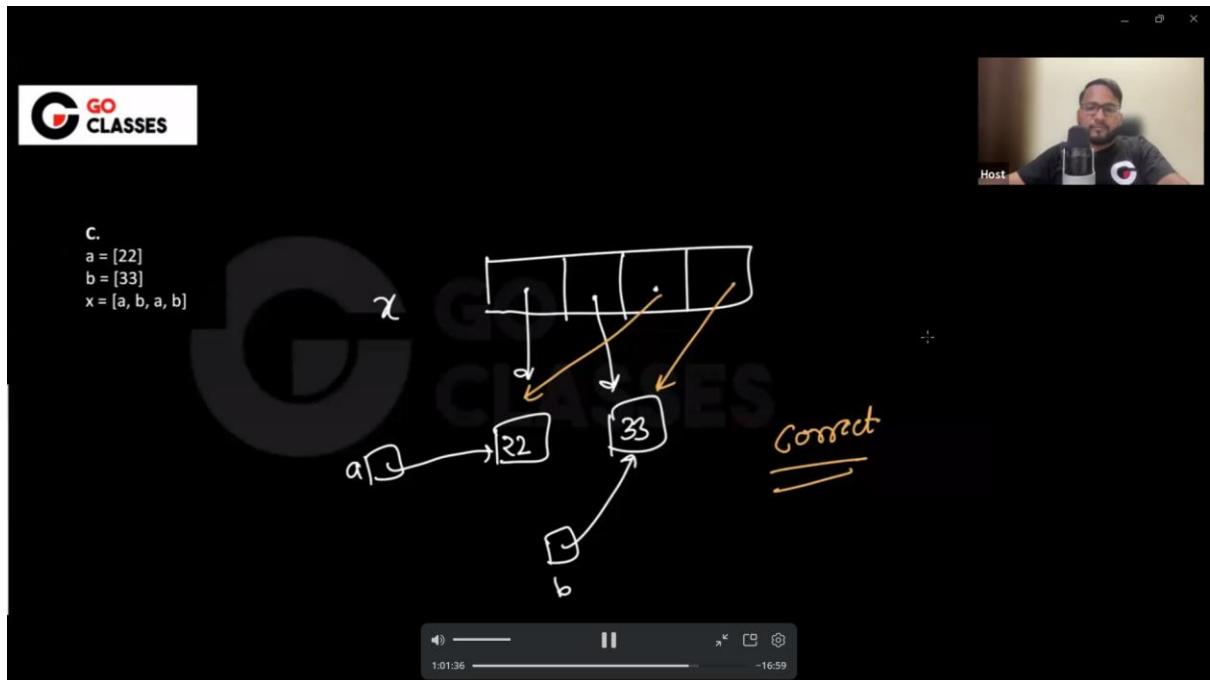
B.  
 $x = [[22], [33]]$   
 $x = x * 2$

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

D.  
 $x = [[22], [33], [22], [33]]$

Host





GO CLASSES

Host

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](https://cs111.wellesley.edu/content/review/Midterm_2_Review_Solutions.pdf)

### Option A

python

Copy code

```
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

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

 Copy code

Now:

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

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

 Copy code

 This exactly matches the given memory diagram!

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

 Copy code

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:]
```

Host

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.

Host

1:08:24 - 10:11



Host

Chrome File Edit View History Bookmarks Profiles Tab Window Help  
localhost:8888/notebooks/Documents/GATE%20DA/Python%20Programming%20GO%20Classes.ipynb  
+  
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 [ ]:



Host

Chrome File Edit View History Bookmarks Profiles Tab Window Help  
localhost:8888/notebooks/Documents/GATE%20DA/Python%20Programming%20GO%20Classes.ipynb  
+  
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 [ ]: |

1:10:11 -08:24

```

>>> 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

```

Host

1:12:04 -06:31

```

>>> 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]]

```

Host

1:12:42 -05:53

**GO CLASSES**

```
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]]

Host

**GO CLASSES**

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

1:14:29 -04:06

Host

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]
```

Host

## Shallow (and Deep) Copies

Host

www.goclasses.in

1:14:37 -03:58



## 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

```
list1 --> ["red", "blue"]
list2 --> [ ]
```

[www.goclasses.in](http://www.goclasses.in)

The screenshot shows a video player interface for a Go Classes lecture. The title "Deep Copy" is displayed at the top. A list of bullet points explains the concept:

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

Below the list is a diagram illustrating the concept of deep copy:

```
graph LR; list1[list1] --> ["red", "blue"]; list2[list2] --> ["red", "yellow"];
```

The diagram shows two boxes labeled "list1" and "list2". Arrows point from each box to a separate box containing a list of items. The first box contains ["red", "blue"] and the second box contains ["red", "yellow"], demonstrating that each variable points to a distinct list.

At the bottom right of the video player, there is a "Host" label next to a small video thumbnail of the instructor.

The screenshot shows a video player interface for a Go Classes lecture. The video frame displays the following Python code:

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

Below the code, a question is asked: "What are the contents of the list **b**?" To the right of the question is a box containing five options:

- 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

At the bottom left of the video player, there is a link: <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

Host

www.goclasses.in

1:15:54 -02:41

## Don't make this mistake

a = [1, 2, 3]  
b = a ← this will Not  
you did not just create a copy of a create a  
new list

Host

www.goclasses.in

1:17:24 -01:11