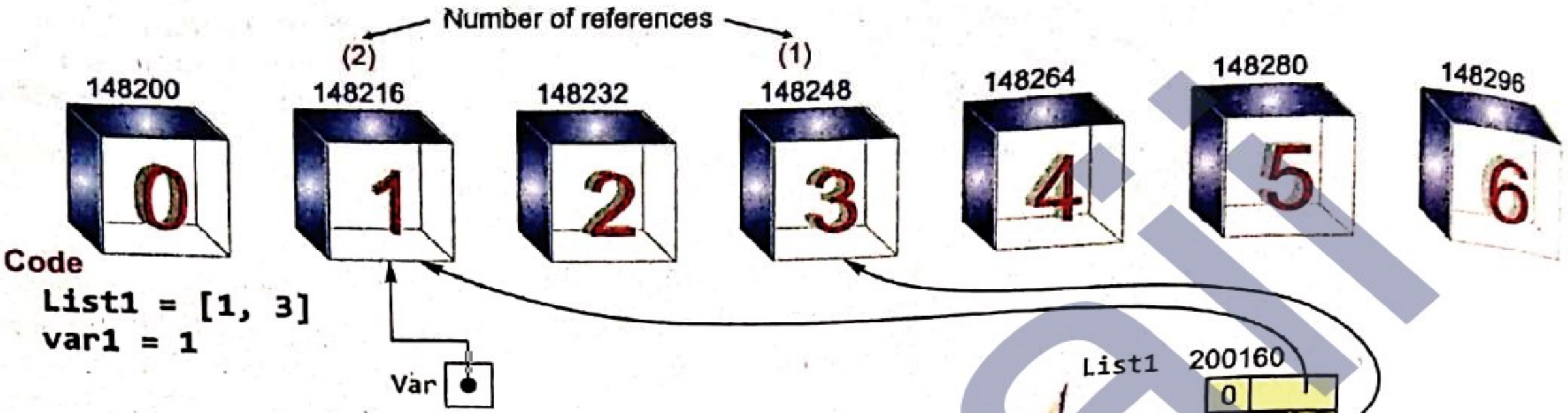
# *Mutable/Immutable Properties of Passed Data Objects*

# *Mutable/Immutable Properties of Passed Data Objects*

# Mutable/Immutable Properties of Passed Data Objects



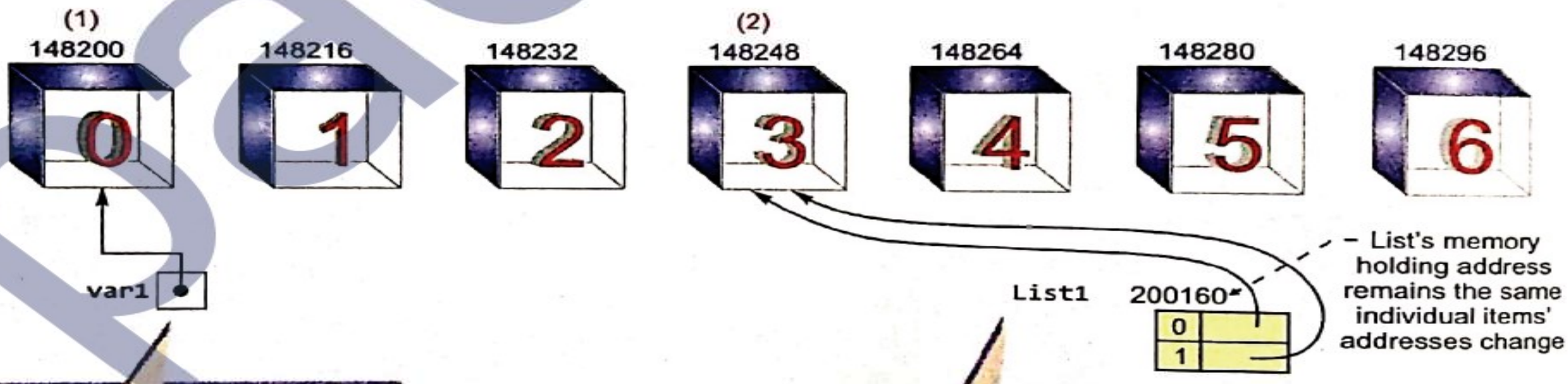

**var1** stores the memory reference of integer value 1 (*i.e.*, 148216) as it is currently storing 1

**List1** has been allocated address 200160 where it can store memory references of its individual items *e.g.*, **List1[0]** currently stores here the memory reference of integer value1 (*i.e.*, 148216) and **List[1]** stores the memory reference of integer value 3 (*i.e.*, 148248)

# Mutable/Immutable Properties of Passed Data Objects

```
var1 = var1 – 1          (Now var1 holds value 0)

List1[0] = 3             (Now first item of List1 is also 3)
```




**var1** now stores memory address of integer 0 (*i.e.*, 148200)

The memory allocated to List1, where it can hold its data items' memory address **remains the same** *i.e.*, it still is 200160, BUT as the individual items of **List1** have changed so individual items' memory addresses stored here have to reflect this. Thus both **List1[0]** and **List1[1]** are currently storing memory address 148248, the memory address of integer 3 as both List1[0] and List1[1] currently hold integer value 3.

# 035 PASSING IMMUTABLE VALUE

```
def myfunc(a):  1 usage
    print("Inside mufunc()")
    print("Value Received in 'a' as", a)
    a = a + 2
    print("Value of 'a' now changes to", a)
    print("Returning from myfunc()")
num = 3
print("Calling myfunc() by passing 'num' with value", num)
myfunc(num)
print("Back from myfunc(). Value of 'num' is", num)
```

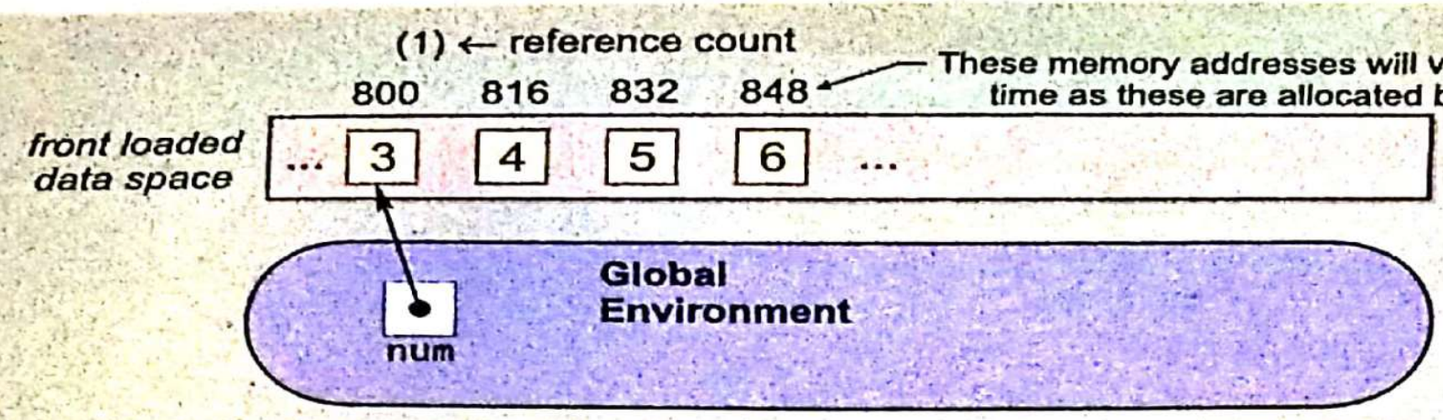
# *Mutable/Immutable Properties of Passed Data Objects*

# Mutable/Immutable Properties of Passed Data Objects

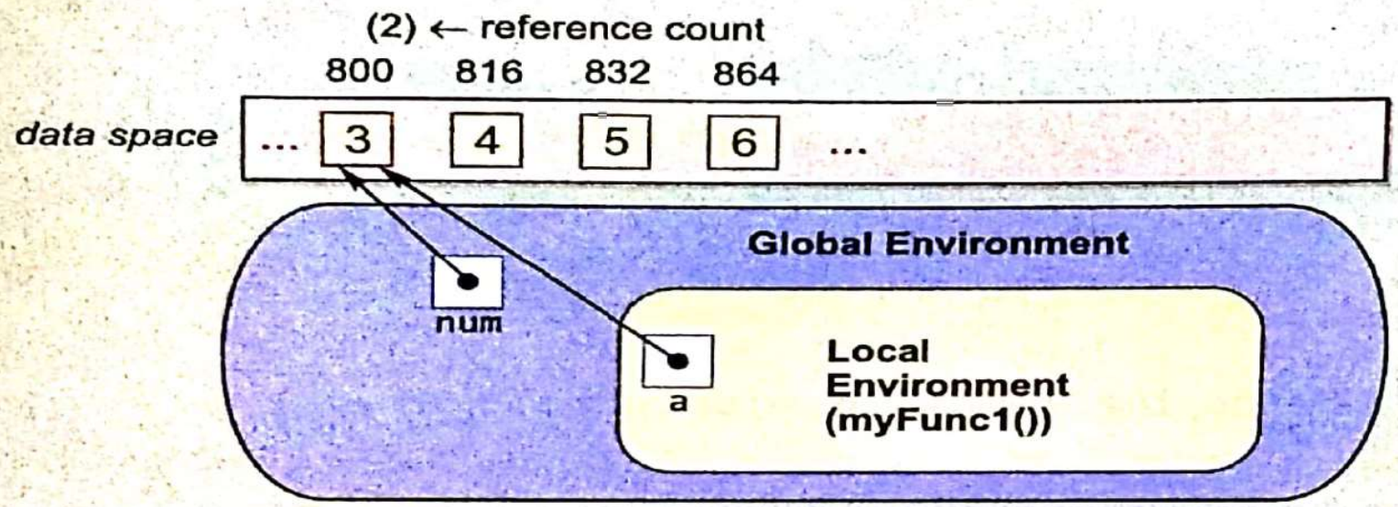
```
Calling myFunc1( ) by passing 'num' with value 3
      Inside myFunc1( )
      Value received in 'a' as 3
      Value of 'a' now changes to (8)
      returning from myFunc1( )
Back from myFunc1( ). Value of 'num' is (3)
```

The value got changed from 3 to 8 inside function
BUT NOT got reflected to __main__

# Mutable/Immutable Properties of Passed Data Objects

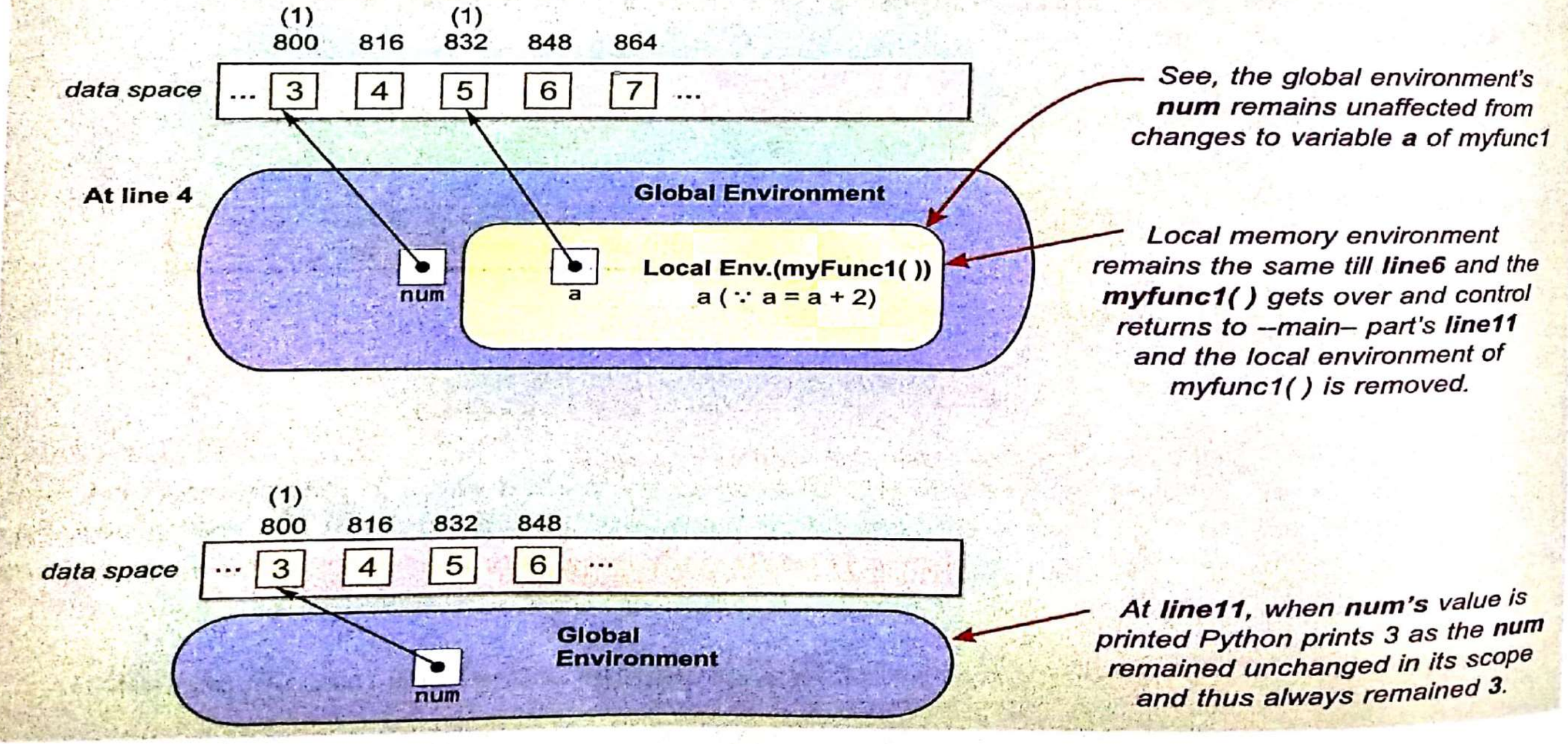



*Till Lines 7-9 of code of –main–*




*At **line10** (function is called) argument num is received in parameter **a** and for **lines 1, 2, 3**, environment remains the same*

# Mutable/Immutable Properties of Passed Data Objects

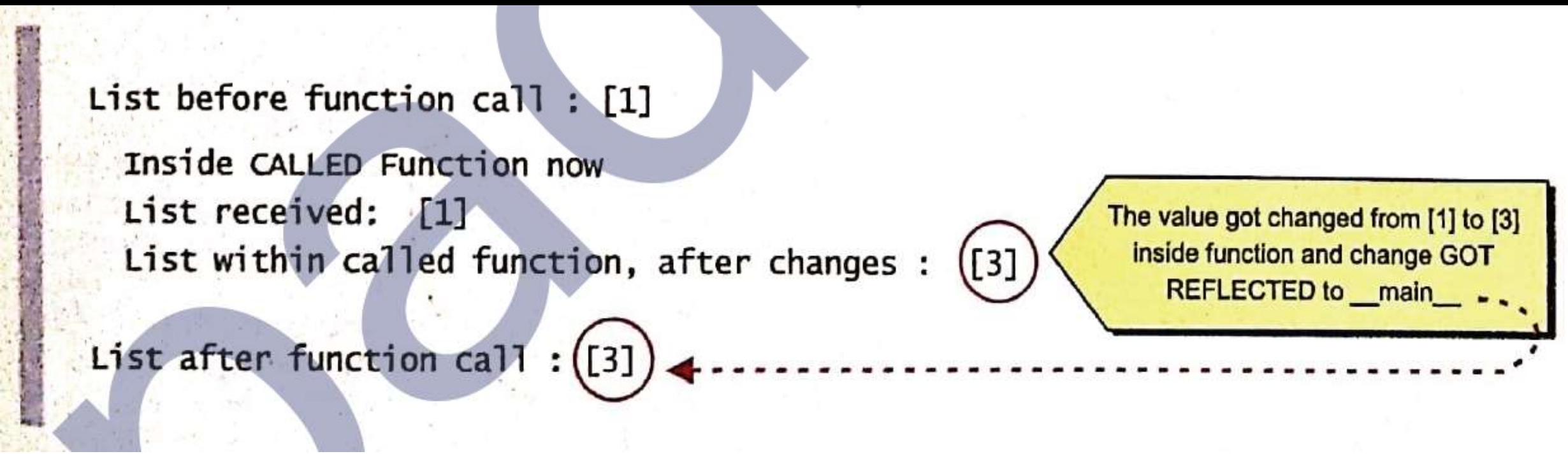


# 036 PASSING MUTABLE VALUE

```
def myfunc2(mylist):  1 usage
    print("Inside CALLED Function now ")
    print("List Received:", mylist)
    mylist[0] += 2
    print("List Within Called function, after changes:", mylist)
list1=[1]
print("List Before Function Call:", list1)
myfunc2(list1)
print("List After Function Call:", list1)
```

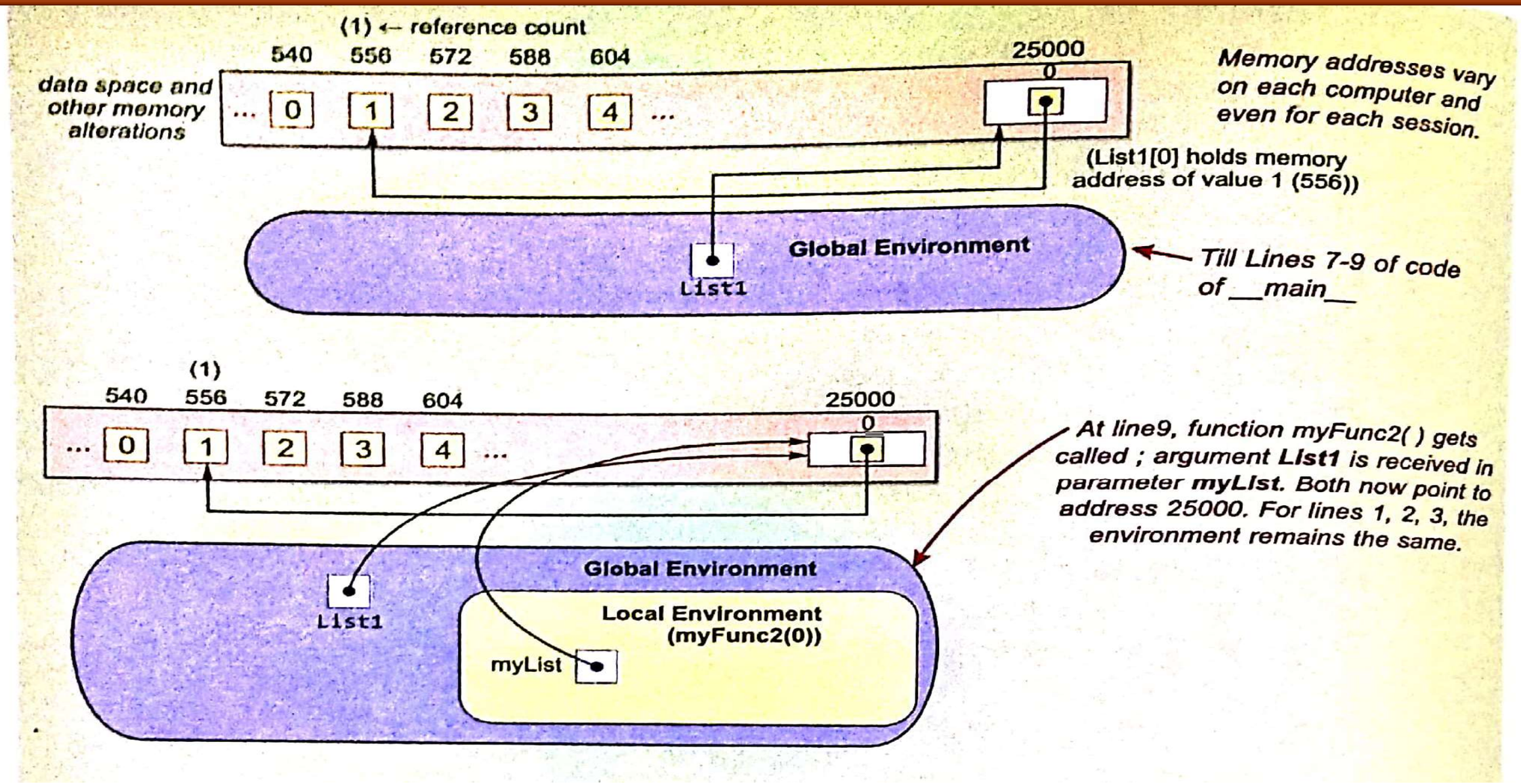
# *Mutable/Immutable Properties of Passed Data Objects*

# Mutable/Immutable Properties of Passed Data Objects

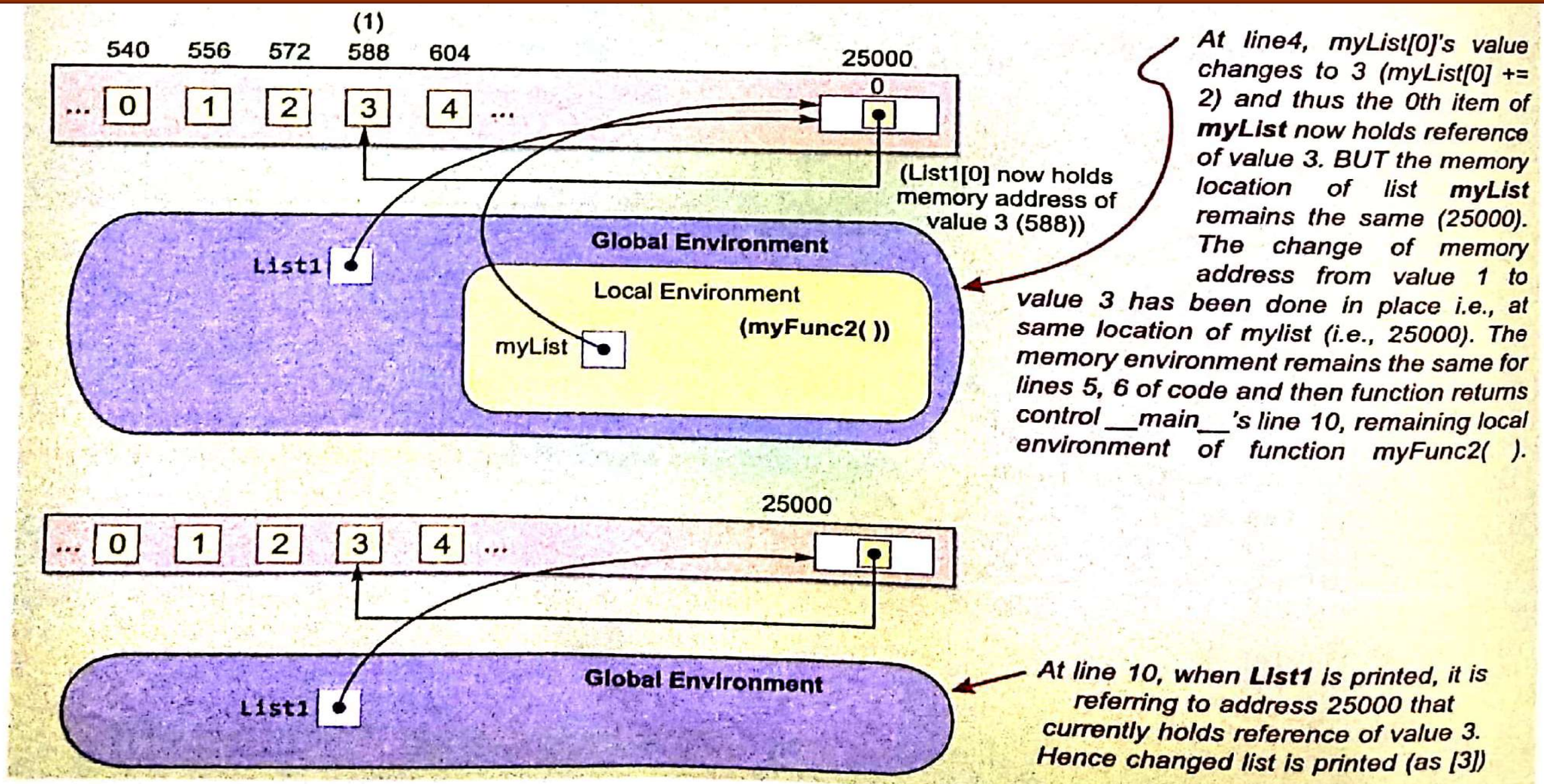


# *Mutable/Immutable Properties of Passed Data Objects*

# Mutable/Immutable Properties of Passed Data Objects



# Mutable/Immutable Properties of Passed Data Objects



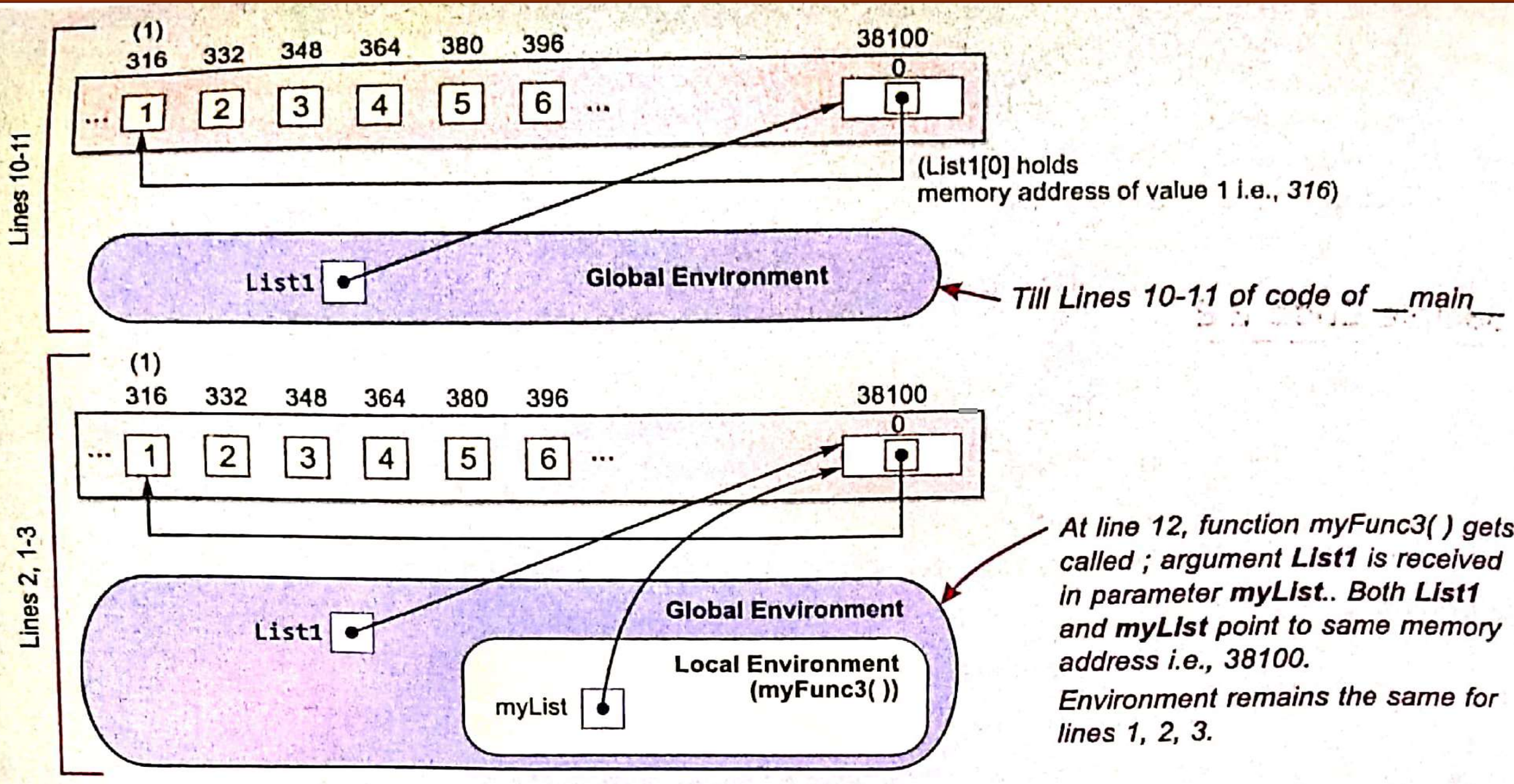
# 037 PASSING MUTABLE VALUE – ADDING DELETING

```
def myfunc3(mylist):  1 usage
    print("Inside CALLED Function now")
    print("List Received:", mylist)
    mylist.append(2)
    mylist.extend([5,1])
    print("List after adding some elements:", mylist)
    mylist.remove(5)
    print("List within called function, after all changes:", mylist)
    return
list1 = [1]
print("List before Function Call: ", list1)
myfunc3(list1)
print("List after Function call: ", list1)
```

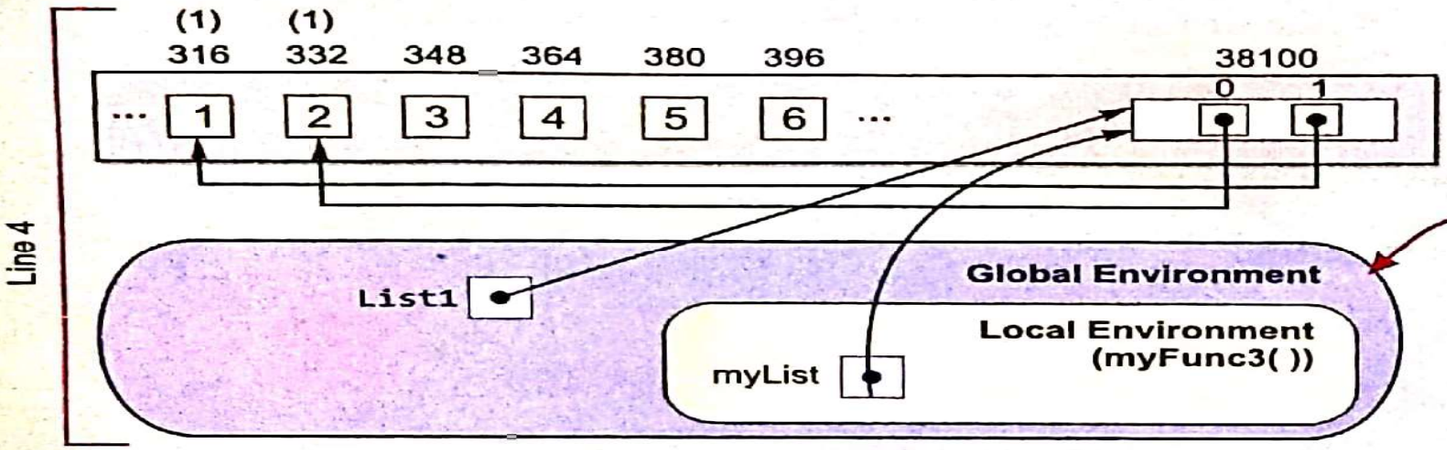
# *Mutable/Immutable Properties of Passed Data Objects*

# *Mutable/Immutable Properties of Passed Data Objects*

# Mutable/Immutable Properties of Passed Data Objects



# Mutable/Immutable Properties of Passed Data Objects



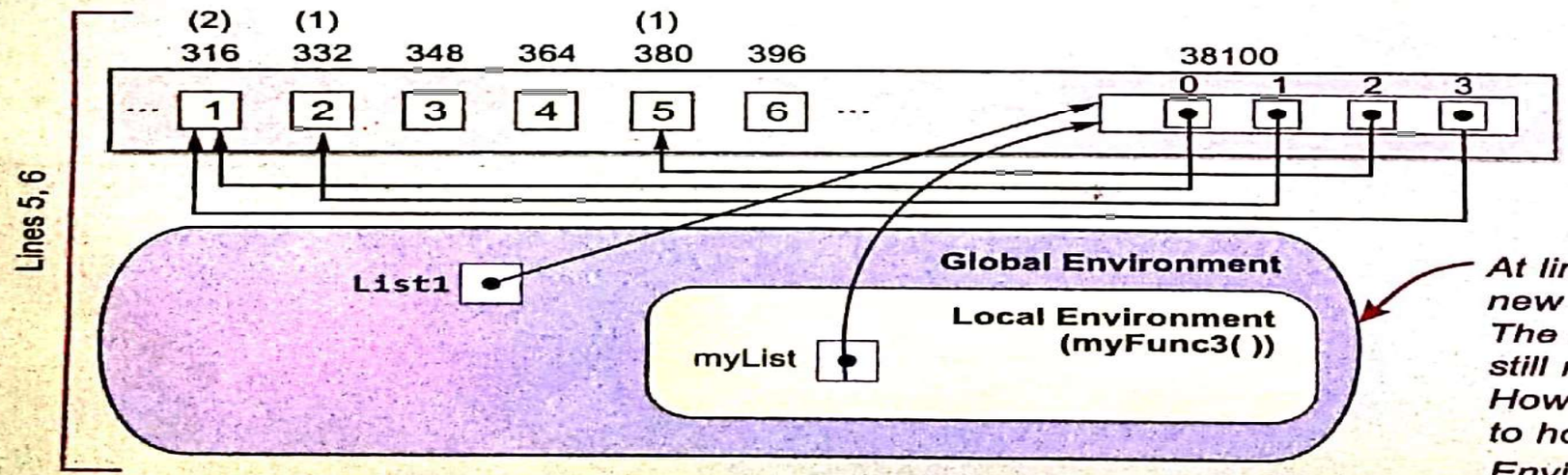

*At line 4, statement **myList.append(2)** gets executed, so a new element (index1) is appended to mylist's same memory address 38100.*

*This new element now references to value2.*

Notice even after appending a new element the list's address is the same *i.e.*, 38100. It, however, now accommodates place to hold new value.

**NOTE**

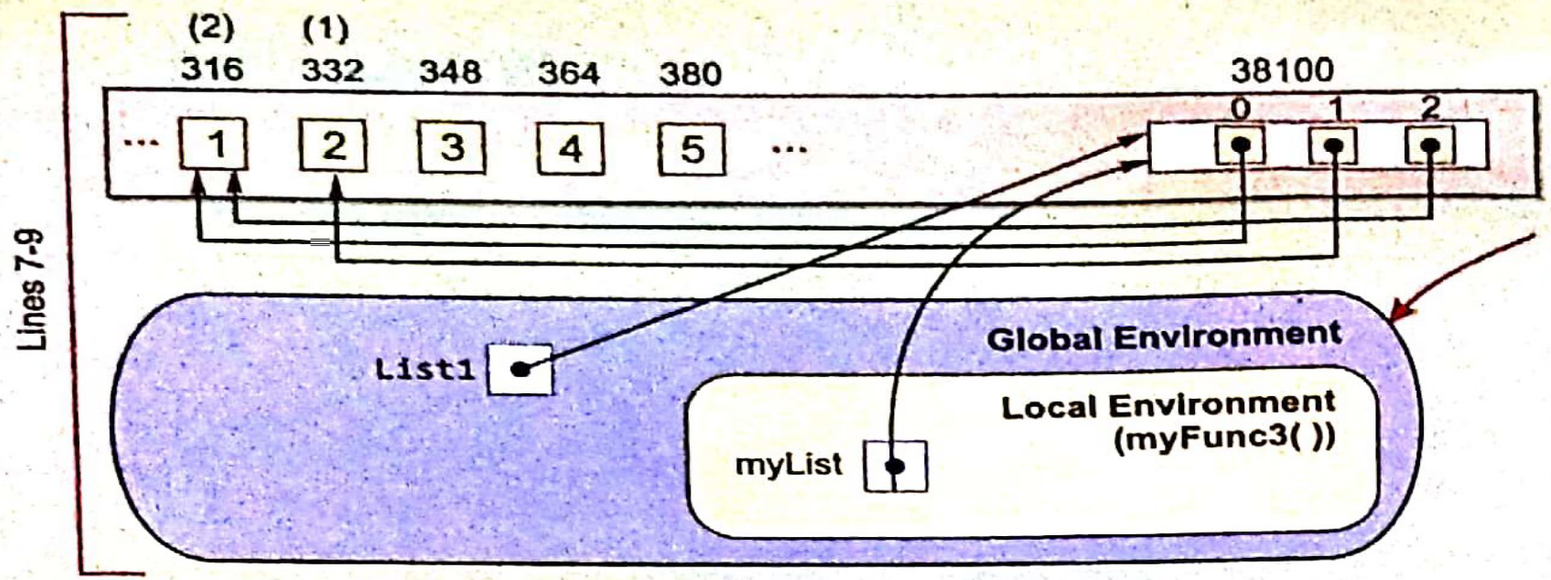
Mutable types act like containers. Container remains unchanged, however, its contents may change. Hence their overall address remains the same as it is the container's address.




*At line5, the list is extended with two new elements (myList.extend(5, 1)). The list's(myList) memory address still remains the same i.e., 38100. However, it now accomodates place to hold two new elements.*

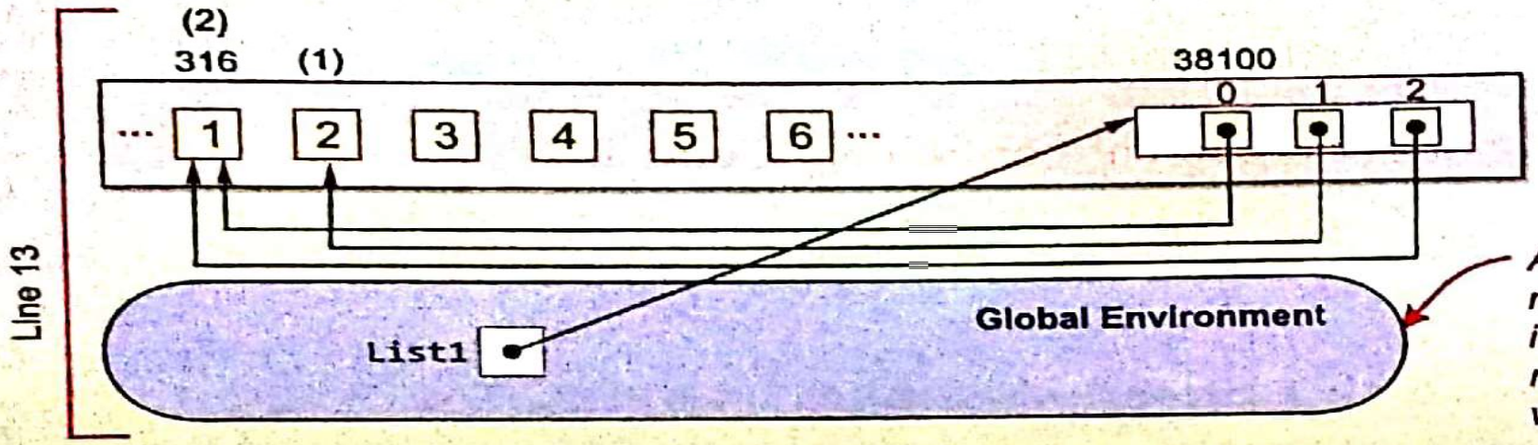
*Environment remains same for line 6 too.*

# Mutable/Immutable Properties of Passed Data Objects



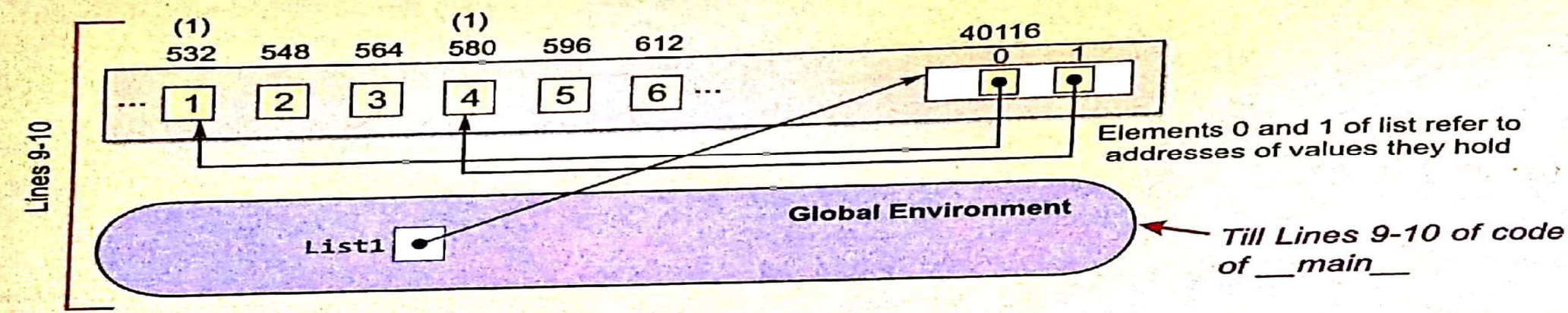

*At line 7, statement **myList.remove(s)** removes element 5 from list and thus only three elements are left in List(**myList**). But the memory address of the list (myList) remains the same even after changes in its contents.*

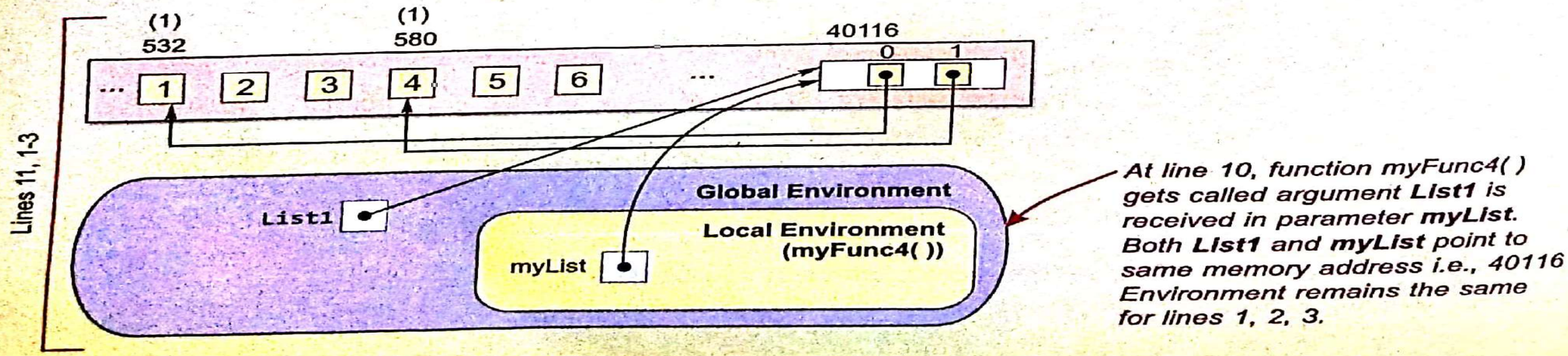
*The environment remains the same till line 9 and then control returns to line13.*



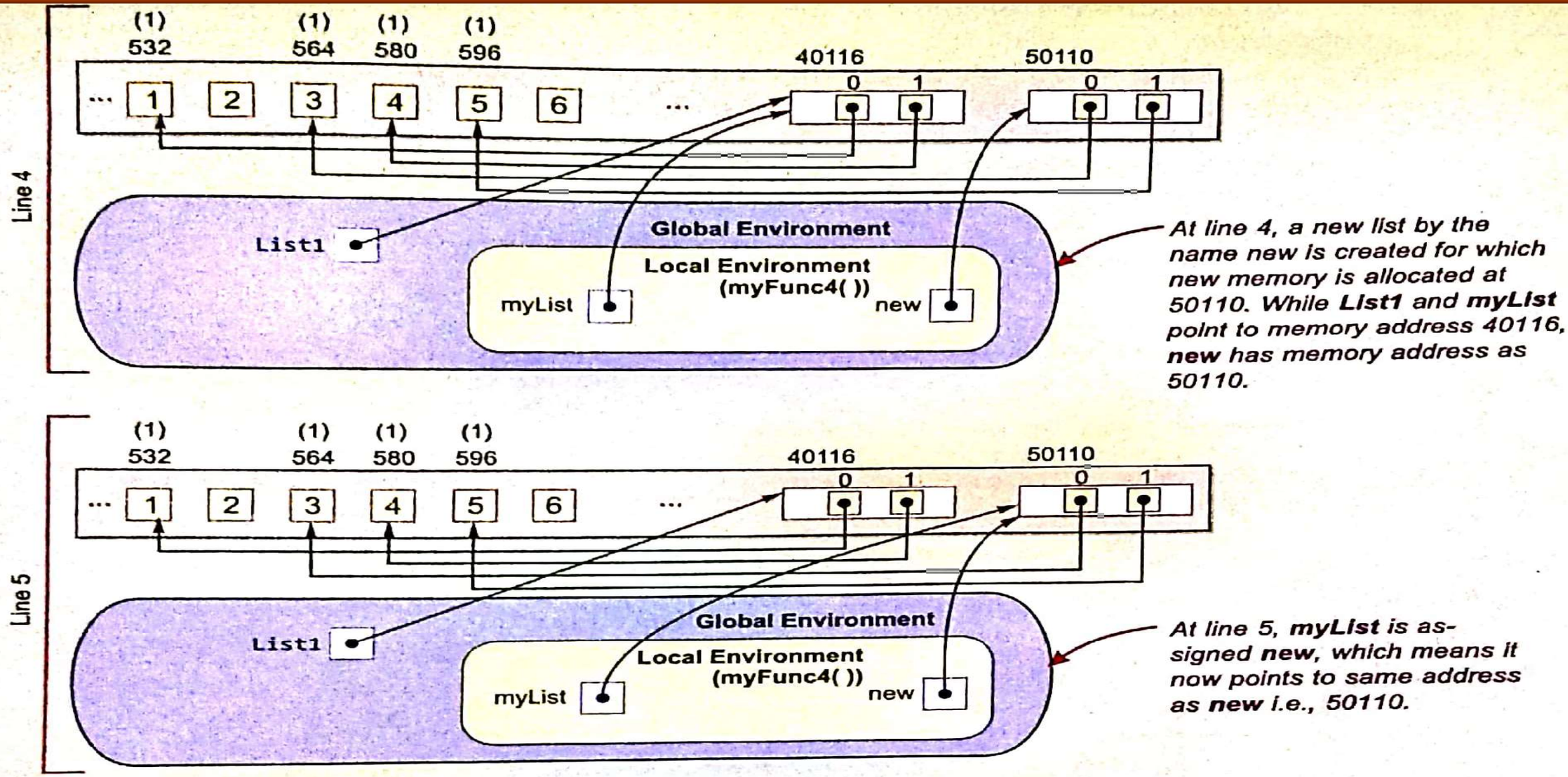

*At line 13, after returning from myFunc3( ) the local environment is removed. The list. List1 is still referencing the address 38100 with all the changes Intact.*

# Mutable/Immutable Properties of Passed Data Objects

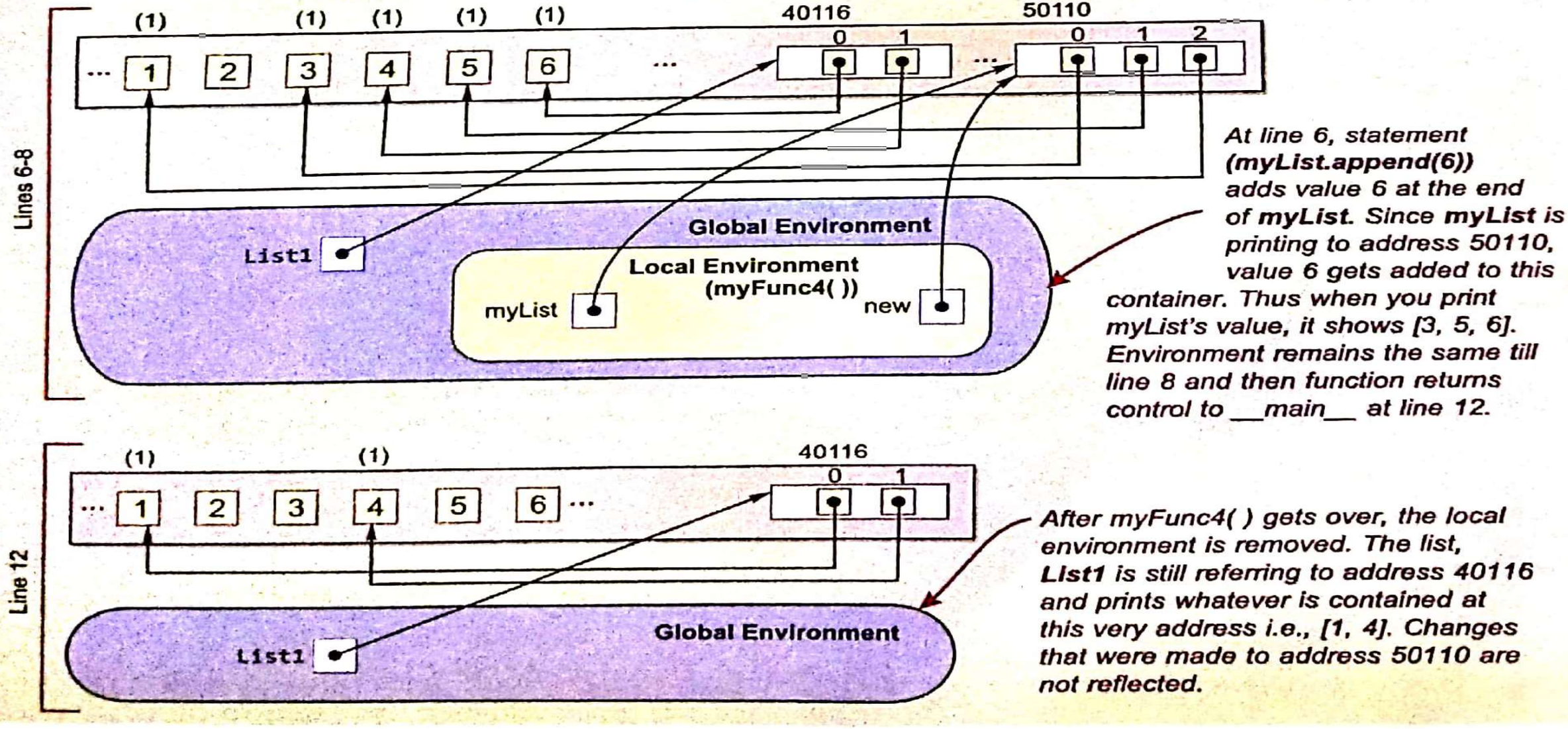


# Mutable/Immutable Properties of Passed Data Objects



# Mutable/Immutable Properties of Passed Data Objects

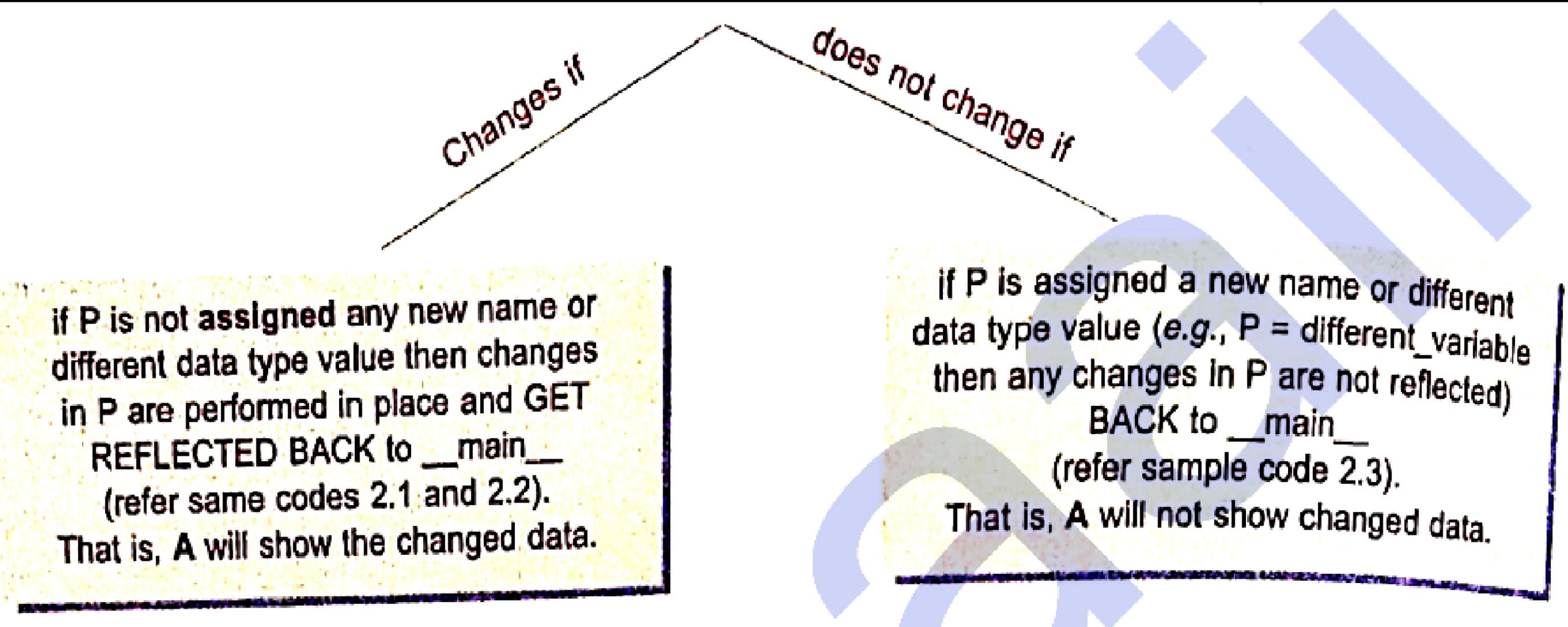


# 038 PASSING MUTABLE VALUE ASSIGNING PARAMETER

```
def myfunc4(mylist):  1 usage
    print("Inside CALLED Function now")
    print("List received:", mylist)
    new = [3,5]
    mylist = new
    mylist.append(6)
    print("List within called function, after changes:", mylist)
    return
list1 = [1,4]
print("List before function call:", list1)
myfunc4(list1)
print("List after function call:", list1)
```

# *Mutable/Immutable Properties of Passed Data Objects*

# Mutable/Immutable Properties of Passed Data Objects



# *Mutable/Immutable Properties of Passed Data Objects*

# *Mutable/Immutable Properties of Passed Data Objects*

# 039 SWAPPING TWO VARIABLES

```
def switch(x, y):  1 usage
    x,y = y,x
    print("Inside Switch:", end =' ')
    print("x = ",x,"y = ",y)
x = 5
y = 7
print("x = ",x,"y = ",y)
switch(x,y)
print("x = ",x,"y = ",y)
```

# 040 CHANGE FROM OCTAL TO OTHERS

```
def oct2others(n):  1 usage
    print("Passed Octal Number: ",n)
    numstring = str(n)
    decnum = int(numstring, 8)
    print("Number in Decimal : ", decnum)
    print("Number in Binary : ",bin(decnum))
    print("Number in Hexadecimal : ",hex(decnum))
num = int(input("Enter an Octal Number: "))
oct2others((num))
```

# 041 GENERATE 4 TERMS OF AP

```
def retseries(init, step):  1 usage
    return init, init+step, init+2*step, init+3*step
ini = int(input("Enter Initial Value of the AP Series: "))
st = int(input("Enter Step Value of the AP Series: "))
print("Series with Initial Value", ini, "& Step Value", st, "Goes as: ")
t1, t2, t3, t4 = retseries(ini, st)
print(t1, end = ", ")
print(t2, end = ", ")
print(t3, end = ", ")
print(t4, end = "...")
```

# 042 CONVERT DOLLAR INTO RUPEES

```python
# Non-void function (returns the converted amount)
def convert_to_rupees(amount_dollars, conversion_rate):  1 usage
    return amount_dollars * conversion_rate

# Void function (prints the converted amount directly)
def print_rupees(amount_dollars, conversion_rate):  1 usage
    rupees = amount_dollars * conversion_rate
    print(f"Converted Amount: ₹{rupees}")
dollars = int(input("Enter Amount in Dollars($): "))
rate = 83.5
# Using non-void function
converted_amount = convert_to_rupees(dollars, rate)
print("Converted Amount (Non-void): ₹", converted_amount)
# Using void function
print_rupees(dollars, rate)
```

# 043 CALCULATE VOLUME OF BOX

```
def volume(length, width, height):  1 usage
    return length * width * height
a=int(input("Enter The Length of the Box: "))
b=int(input("Enter The Width of the Box: "))
c=int(input("Enter The height of the Box: "))
print("The volume of box is", volume(a, b, c))
```

# 044 CALCULATE CUBE

```
def cube(x):
    return x ** 3
a=int(input("Enter a Number : "))
print("The Cube of",a, "is", cube(a))
```

# 045 RANDOM NO.

```
import random
def random(start, end):  2 usages
    return random.randint(start, end)
start = int(input("Enter the starting number of range: "))
end = int(input("Enter the ending number of range: "))
print("Three Random Numbers:")
for i in range(3):
    print(random(start, end))
```

# 046 LENGTH OF STRING

```python
def same_length(str1, str2):  1 usage
    return len(str1) == len(str2)
s1 = input("Enter first string: ")
s2 = input("Enter second string: ")
if same_length(s1, s2):
    print("✅ Both strings are of the same length.")
else:
    print("❌ Strings are of different lengths.")
```

# 047 COMPARISON OF TWO NUMBERS

```python
def min_ones_digit(num1, num2):  1 usage
    ones1 = num1 % 10
    ones2 = num2 % 10
    if ones1 < ones2:
        return num1
    elif ones2 < ones1:
        return num2
    else:
        return "Both numbers have the same one's digit."
num1 = int(input("Enter first number: "))
num2 = int(input("Enter second number: "))
result = min_ones_digit(num1, num2)
print("Number with minimum one's digit is:", result)
```

# 048

```python
def smaller_number(num1, num2):  1 usage
    if num1 < num2:
        return num1
    elif num2 < num1:
        return num2
    else:
        return "Both numbers are equal."
num1 = int(input("Enter first number: "))
num2 = int(input("Enter second number: "))
result = smaller_number(num1, num2)
print("The smaller number is:", result)
```