**Shallow vs Deep Deep Copy in List**

The expression **two\_d\_list = [[0] \* col] \* row** in Python creates a list containing **row** number of references to the same list (**[0] \* col**). This means that all rows in **two\_d\_list** are actually references to the same list in memory. As a result, if you modify one row, you end up modifying all rows, because they are all the same list.

# **Why It Creates a Shallow Copy**

* **List Multiplication**: When you multiply a list by a number (like **[0] \* col**), it creates a new list by repeating the elements. However, when you multiply a list of lists (like **[[0] \* col] \* row**), it repeats the list reference, not the actual list.
* **Single Object in Memory**: All **row** elements of **two\_d\_list** point to the same list object in memory. They are not independent lists.

# **Proper Way to Create a 2D List**

To create a **row** x **col** grid where each row is independent, you should use a list comprehension. This ensures that each row is a separate list in memory.

two\_d\_list = [[0 for \_ in range(col)] for \_ in range(row)]

This expression creates a new list for each row. The inner list comprehension **[0 for \_ in range(col)]** creates a list with **col** number of zeros. The outer list comprehension is executed **row** times to create **row** number of independent lists.

**Example**

row = 3

col = 4

# Correct way to create a 2D list

two\_d\_list = [[0 for \_ in range(col)] for \_ in range(row)]

# Modifying one element

two\_d\_list[0][0] = 1

print(two\_d\_list)  # Output: [[1, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]

**Code Snippet with Shallow Copy**

row = 3

col = 4

# Creating a 2D list using your method

two\_d\_list = [[0] \* col] \* row

# Let's modify one element

two\_d\_list[0][0] = 1

print(two\_d\_list)

**Expected Output and Explanation**

The expected output of this code is:

[[1, 0, 0, 0], [1, 0, 0, 0], [1, 0, 0, 0]]

This output might be surprising if you're expecting only the first element of the first row to change. However, since **two\_d\_list** is created using a shallow copy method, all rows are references to the same list. Therefore, modifying an element in one row actually modifies that element in all rows.

# **Why This Happens**

* **[[0] \* col] \* row**: This creates a list with **row** references to the same list (**[0] \* col**). It doesn't create **row** independent lists.
* **Modification Propagation**: When you modify **two\_d\_list[0][0]**, you're modifying the first element of the list that all rows reference. Hence, the change appears in each row.

This behavior is a direct result of how the list multiplication works in Python and is a classic example of a shallow copy in a 2D list context.

To properly initialize a two-dimensional list, we must ensure that each cell of the primary list refers to an *independent* instance of a secondary list. This can be accomplished through the use of Python's list comprehension syntax.

data = [ [0] \* c for j in range(r)]

This command produces a valid configuration. By using list comprehension, the expression **[0] \* c** is reevaluated for each pass of the embedded for loop. Therefore, we get *r* distinct secondary lists, as desired. (We note that the variable *j* in that command is irrelevant; we simply need a for loop that iterates *r* times.)

data = []

for i in range(row):

    sublist = []

    for j in range(column):

        sublist.append(0)

    data.append(sublist)