**AI ASSISTED CODING**

**LAB EXAM-2**

**A.DHANALAXMI**

**2403A51269**

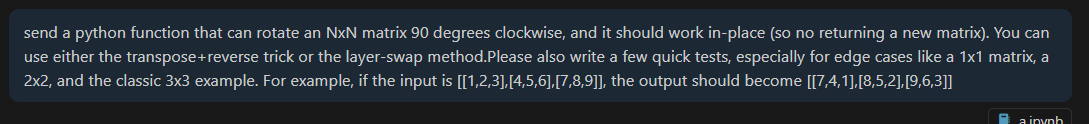
**Batch-12**

**SET-K**

**Subgroup K:**

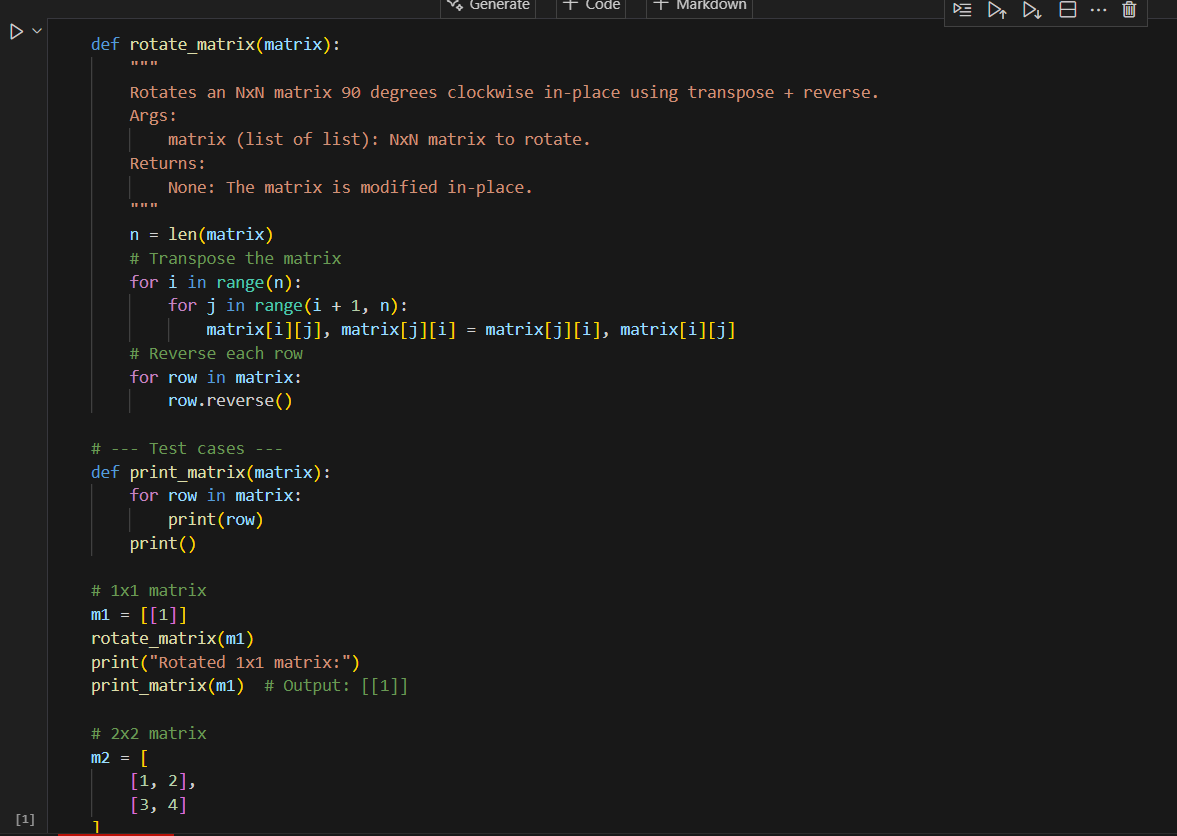
**K.1 — [S18K1] Rotate NxN matrix 90° clockwise:**

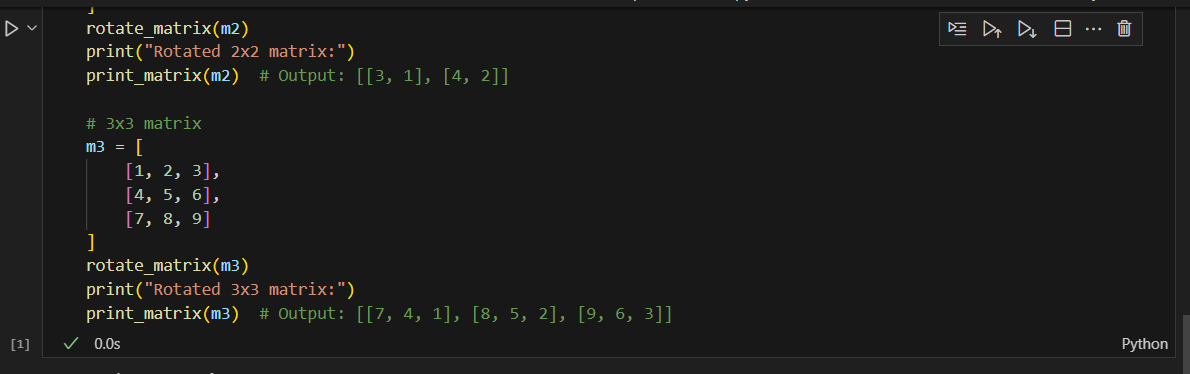
**Prompt used:**



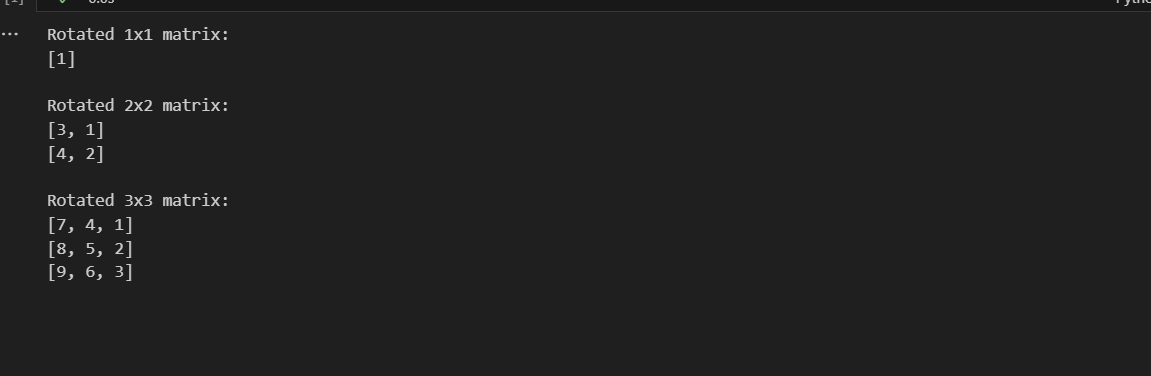
**Response :**

**Code :**





**Output:**

****

**Code with line-by-line explanation:**

def rotate\_matrix(matrix):

"""

Rotates an NxN matrix 90 degrees clockwise in-place using transpose + reverse.

Args:

matrix (list of list): NxN matrix to rotate.

Returns:

None: The matrix is modified in-place.

"""

👉 This defines the function rotate\_matrix which takes a square matrix (same number of rows and columns).  
The function doesn’t return anything, it just modifies the matrix directly.

n = len(matrix)

👉 n stores the size of the matrix (number of rows/columns).  
For example, if matrix = [[1,2,3],[4,5,6],[7,8,9]], then n = 3.

# Transpose the matrix

for i in range(n):

for j in range(i + 1, n):

matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i][j]

👉 This transposes the matrix.

* Transpose means converting rows into columns.
* Example:
* [1,2,3] [1,4,7]
* [4,5,6] → [2,5,8]
* [7,8,9] [3,6,9]
* The nested loop swaps matrix[i][j] with matrix[j][i] for all elements above the diagonal.
* for j in range(i + 1, n): ensures we don’t swap elements twice.

# Reverse each row

for row in matrix:

row.reverse()

👉 After transpose, we reverse each row to complete 90° clockwise rotation.

* Example (after transpose):
* [1,4,7]
* [2,5,8]
* [3,6,9]
* After reversing each row:
* [7,4,1]
* [8,5,2]
* [9,6,3]

✔ Final result → rotated matrix by 90° clockwise.

Helper Function

def print\_matrix(matrix):

for row in matrix:

print(row)

print()

👉 This function just prints the matrix row by row for easy viewing.  
The print() at the end adds a blank line for spacing.

Test Cases

# 1x1 matrix

m1 = [[1]]

rotate\_matrix(m1)

print("Rotated 1x1 matrix:")

print\_matrix(m1) # Output: [[1]]

👉 A single element matrix remains the same after rotation.

# 2x2 matrix

m2 = [

[1, 2],

[3, 4]

]

rotate\_matrix(m2)

print("Rotated 2x2 matrix:")

print\_matrix(m2) # Output: [[3, 1], [4, 2]]

👉 Step by step:

* Original:
* [1,2]
* [3,4]
* Transpose:
* [1,3]
* [2,4]
* Reverse rows:
* [3,1]
* [4,2]

# 3x3 matrix

m3 = [

[1, 2, 3],

[4, 5, 6],

[7, 8, 9]

]

rotate\_matrix(m3)

print("Rotated 3x3 matrix:")

print\_matrix(m3) # Output: [[7, 4, 1], [8, 5, 2], [9, 6, 3]]

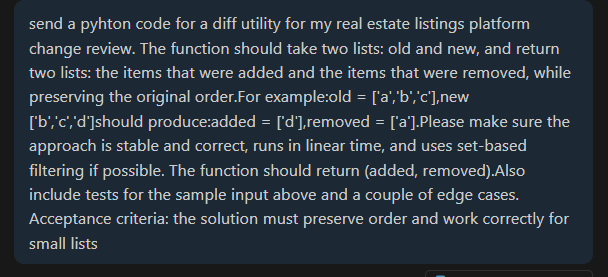
👉 Step by step:

* Original:
* [1,2,3]
* [4,5,6]
* [7,8,9]
* Transpose:
* [1,4,7]
* [2,5,8]
* [3,6,9]
* Reverse rows:
* [7,4,1]
* [8,5,2]
* [9,6,3]

✅ Summary in one line:  
The code works by transposing the matrix (rows → columns) and then reversing each row, which rotates the matrix by 90° clockwise

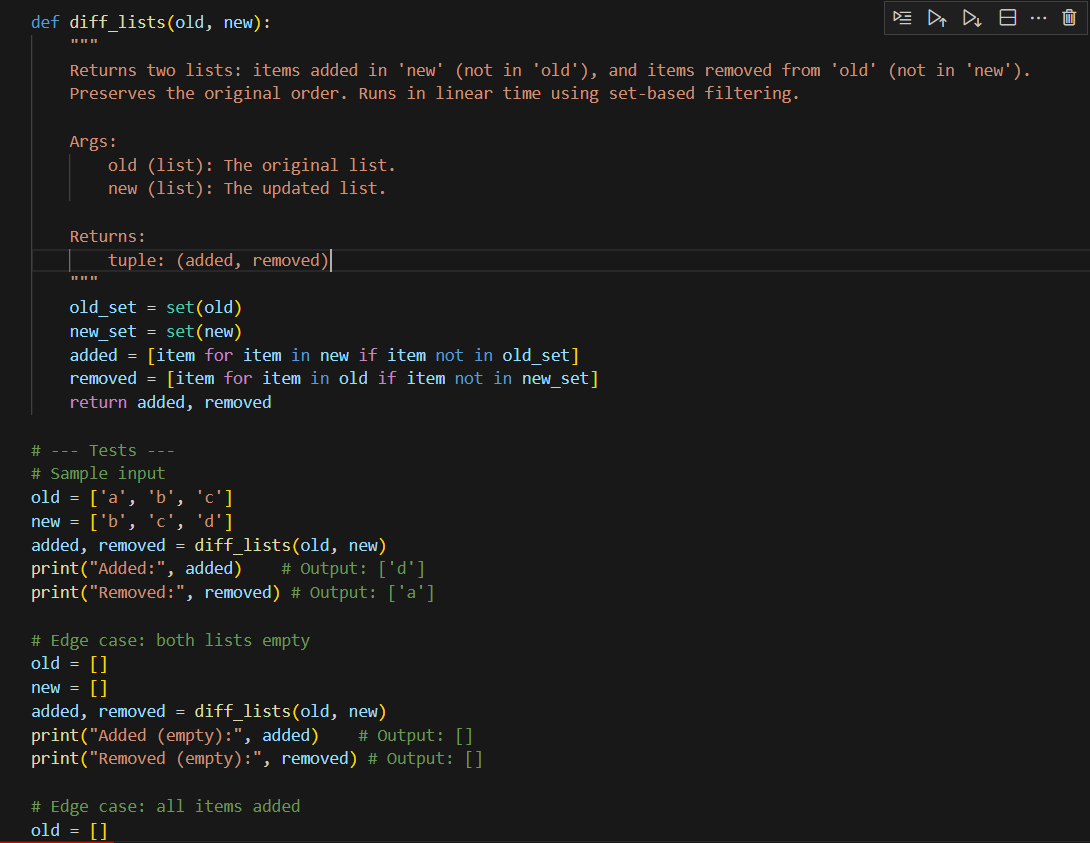
**K.2 — [S18K2] Compute added/removed lines**

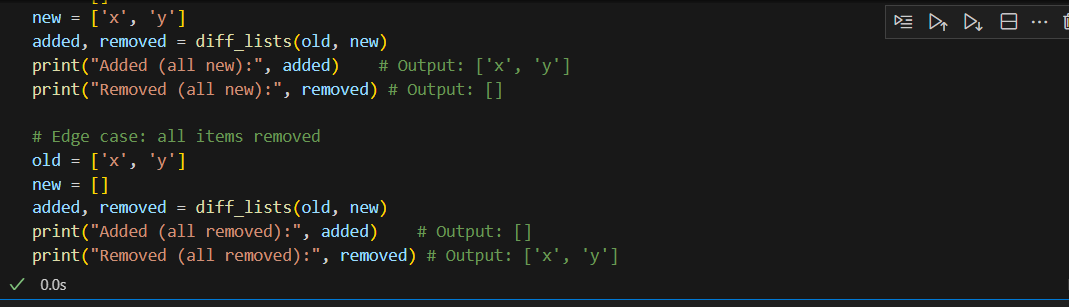
**Prompt used:**



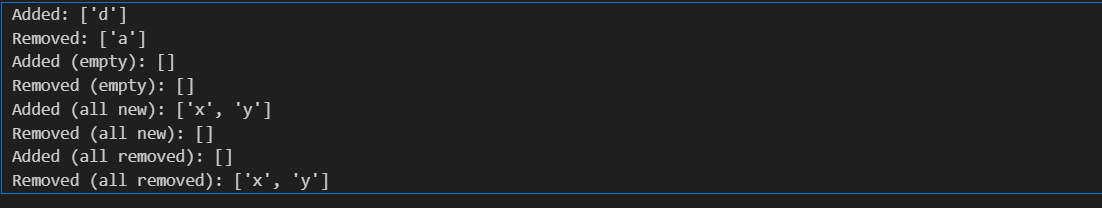
**Response:**

**Code:**





**Output:**



**Code Explanation (Line by Line):**

def diff\_lists(old, new):

👉 Defines a function diff\_lists with parameters:

* old = original list
* new = updated list

"""

Returns two lists: items added in 'new' (not in 'old'), and items removed from 'old' (not in 'new').

Preserves the original order. Runs in linear time using set-based filtering.

"""

👉 Docstring describing what the function does.

old\_set = set(old)

new\_set = set(new)

👉 Converts old and new into sets for **fast lookup (O(1))**.

* old\_set → unique elements from old
* new\_set → unique elements from new

added = [item for item in new if item not in old\_set]

👉 Creates a list added containing all items that:

* Appear in new
* But are **not** in old\_set
* Keeps the order because we loop over new

Example: new=['b','c','d'], old\_set={'a','b','c'} → only 'd' is new.

removed = [item for item in old if item not in new\_set]

👉 Creates a list removed containing all items that:

* Appear in old
* But are **not** in new\_set
* Keeps the order because we loop over old

Example: old=['a','b','c'], new\_set={'b','c','d'} → 'a' is missing in new.

return added, removed

👉 Returns both lists as a tuple (added, removed).

**🔎 Dry Run (Example)**

Input:

old = ['a','b','c']

new = ['b','c','d']

1. old\_set = {'a','b','c'}  
   new\_set = {'b','c','d'}
2. Loop over new=['b','c','d']:
   * 'b' in old\_set → skip
   * 'c' in old\_set → skip
   * 'd' not in old\_set → add 'd'  
     → added=['d']
3. Loop over old=['a','b','c']:
   * 'a' not in new\_set → add 'a'
   * 'b' in new\_set → skip
   * 'c' in new\_set → skip  
     → removed=['a']
4. Final result:
5. added=['d'], removed=['a']

✅ So your **output is correct**:

Added: ['d']

Removed: ['a']