

user

```
matrix = [[1,2,3,4], [5,6,7,8],  
[9,10,11,12]]
```

new_matrix = [[row[i] for row in matrix] for i in range(4)] 这个矩阵置换我不是特别明白能不能很详细的帮我讲解一下

assistant

当然可以！我们一步一步来详细讲解这段代码是如何实现矩阵转置（transpose）的。

📌 原始矩阵

你给出的 `matrix` 是一个 3 行 4 列的二维列表（也就是 3×4 矩阵）：

```
matrix = [  
    [1, 2, 3, 4], # 第0行  
    [5, 6, 7, 8], # 第1行  
    [9,10,11,12] # 第2行  
]
```

它的结构是：

- 行索引：0, 1, 2（共3行）
- 列索引：0, 1, 2, 3（每行有4个元素）

📌 目标：矩阵转置

转置的意思是：把原来的“行”变成“列”，“列”变成“行”。

所以，转置后的矩阵应该是 4 行 3 列，如下所示：

```
原 matrix 的第0列 → 新矩阵的第0行: [1, 5, 9]  
原 matrix 的第1列 → 新矩阵的第1行: [2, 6, 10]  
原 matrix 的第2列 → 新矩阵的第2行: [3, 7, 11]  
原 matrix 的第3列 → 新矩阵的第3行: [4, 8, 12]
```

最终结果应为：

```
[  
    [1, 5, 9],  
    [2, 6, 10],  
    [3, 7, 11],
```

📌 代码解析

现在来看这行代码：

```
new_matrix = [[row[i] for row in matrix] for i in range(4)]
```

这是一个嵌套的列表推导式（nested list comprehension）。我们从外到内拆解：

- ◆ 外层循环：`for i in range(4)`
 - `range(4)` 表示 $i = 0, 1, 2, 3$
 - 每次 i 对应原始矩阵的一列（因为原始矩阵有4列）
 - 所以，外层循环在遍历每一列的索引
- ◆ 内层表达式：`[row[i] for row in matrix]`
 - 对于固定的 i （比如 $i=0$ ），我们遍历 `matrix` 中的每一行（`row`）
 - 然后取该行的第 i 个元素：`row[i]`
 - 把所有行的第 i 个元素收集起来，就构成了新矩阵的第 i 行

✓ 举例说明：

当 $i = 0$ ：

- `row = [1,2,3,4] → row[0] = 1`
- `row = [5,6,7,8] → row[0] = 5`
- `row = [9,10,11,12] → row[0] = 9`
- 所以内层结果是 `[1, 5, 9] → 这就是转置后的第0行`

当 $i = 1$ ：

- 取每行的第1个元素：`2, 6, 10 → [2, 6, 10]`

以此类推……

🧠 总结逻辑

这个列表推导式的思路是：

“对于每一列索引 i （0 到 3），把所有行在该列上的元素提取出来，组成一个新行。”

这就完成了按列取值 → 变成新行的过程，即转置。

补充：更通用的写法

如果你不知道列数（比如矩阵大小不固定），可以用：

```
new_matrix = [[row[i] for row in matrix] for i in range(len(matrix[0]))]
```

因为 `len(matrix[0])` 就是第一行的长度，也就是列数。

或者用 `zip` 实现更简洁的转置：

```
new_matrix = list(zip(*matrix))
```

但注意：`zip` 返回的是元组，不是列表，如果需要纯列表，可以：

```
new_matrix = [list(col) for col in zip(*matrix)]
```

最终输出验证

运行你的代码：

```
matrix = [
    [1,2,3,4],
    [5,6,7,8],
    [9,10,11,12]
]

new_matrix = [[row[i] for row in matrix] for i in range(4)]
print(new_matrix)
```

输出：

```
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

完全正确！

如有任何一步还不清楚，欢迎继续追问 😊