

DAY - 8 : Deep Dive into Python Lists

Content overview :

- Looping Through lists.
 - Nested lists.
 - Nested list Comprehension.
-

Looping through lists:

1. Using "for" loop :

The most straightforward way to iterate through a list.

```
fruits = ["apple", "banana", "mango", "orange"]

for fruit in fruits:
    print(fruit)
```

Output:

```
apple
banana
mango
orange
```

3. Using Range and Index :

Access both index and value using `range()` and `len()`.

```
fruits = ["apple", "banana", "mango", "orange"]

for i in range(len(fruits)):
```

```
    print(i, fruits[i])
```
```

\*\*Output:\*\*

```
```

```

```
0 apple
1 banana
2 mango
3 orange
```

4. Using "While" loop :

Traditional loop approach with manual index management.

```
numbers = [10, 20, 30, 40]
i = 0
```

```
while i < len(numbers):
    print(numbers[i])
    i += 1
```

```

\*\*Output:\*\*

```
```

```

```
10
20
30
40
```

4. Using enumerate() :

Get both index and value elegantly. You can also customize the starting index.

```
names = ["dermin", "derminotsocool", "derminiscool"]
```

```
for index, name in enumerate(names, start=1):
```

```
    print(index, name)
```

```

\*\*Output:\*\*

```
```

```

```
1 dermin
2 derminotsocool
3 derminiscool

```

5. List Comprehension :

Create new lists in a single, readable line.

```
squares = [x*x for x in [1, 2, 3, 4]]
print(squares)
```

```

\*\*Output:\*\*

```
```

```

```
[1, 4, 9, 16]

```

Nested Lists:

Lists that contain other lists as elements.

```
nested = [[1, 2, 3], [4, 5, 6]]
```

```
# nested[0] → [1, 2, 3]
```

```
# nested[1] → [4, 5, 6]
```

Accessing Values in Nested Lists :

Use multiple indices to access elements.

```
nested = [[1, 2, 3], [4, 5, 6]]  
print(nested[1][2])
```

BREAKDOWN:

- `nested[1]` → second list → `[4, 5, 6]`
- `[2]` → value at index 2 → `6`

Output: `6`

Looping Through Nested Lists:

Example: Student Records

```
students = [  
    ["Alice", 85, 92],  
    ["Bob", 78, 88],  
    ["Charlie", 95, 89]  
]  
  
for i, student in enumerate(students):  
    print(f"Student {i+1}:")  
    for j, value in enumerate(student):  
        if j == 0:  
            print(f"  Name: {value}")  
        else:  
            print(f"  Score {j}: {value}")  
    print()  
...  
  
**Output:**  
...  
Student 1:  
  Name: Alice  
  Score 1: 85  
  Score 2: 92
```

Student 2:

Name: Bob

Score 1: 78

Score 2: 88

Student 3:

Name: Charlie

Score 1: 95

Score 2: 89

Example: Sales Data Analysis

```
sales_data = [
    [100, 200, 300],
    [150, 250, 350],
    [175, 275, 375]
]

# Weekly totals
for i, week in enumerate(sales_data):
    total = sum(week)
    print(f"Week {i+1} total: ${total}")

# Daily totals across all weeks
for day in range(len(sales_data[0])):
    total = sum(week[day] for week in sales_data)
    print(f"Day {day+1} total: ${total}")
```

Output:
```
Week 1 total: $600
Week 2 total: $750
Week 3 total: $825
Day 1 total: $425

```

```
Day 2 total: $725
Day 3 total: $1025
```

Nested List Comprehension:

Flattening a Nested List -

Convert a 2D list into a 1D list.

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

flat = [value for row in matrix for value in row]
print(flat)
```

Output: [1, 2, 3, 4, 5, 6, 7, 8, 9]

Transforming While Flattening -

Apply operations while flattening.

```
squares = [num*num for row in matrix for num in row]
print(squares)
```

Output: [1, 4, 9, 16, 25, 36, 49, 64, 81]

Creating a 2D List -

Keep the structure but transform values.

```
multiplied = [[num * 10 for num in row] for row in matrix]
print(multiplied)
```

Output: [[10, 20, 30], [40, 50, 60], [70, 80, 90]]

Transposing a Matrix -

Swap rows and columns.

```

matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

transpose = [[matrix[r][c] for r in range(len(matrix))]
             for c in range(len(matrix[0]))]
print(transpose)

```

Output:

```

[[1, 4, 7],
 [2, 5, 8],
 [3, 6, 9]]

```

Irregular Nested Lists -

Nested lists don't need to have the same length.

```
weird = [[1, 2, 3], [4], [5, 6, 7, 8, 9]]
```

This is a valid nested list where each sub-list has a different number of elements.

Summary:

- ✓ Multiple ways to loop through lists: `for`, `while`, `enumerate()`, list comprehension
- ✓ Nested lists allow for matrix-like data structures
- ✓ Use double indexing to access nested list elements: `list[row][column]`
- ✓ List comprehension works with nested lists for powerful one-liners
- ✓ Transposing matrices is possible with nested comprehension
- ✓ Nested lists can have irregular shapes