Python DSA – Pattern & Template Handbook (Beginner → Advanced)

यह डॉक्यूमेंट Data Structures & Algorithms (DSA) के **सबसे ज़्यादा इस्तेमाल होने वाले patterns और templates** को एक जगह concise और practical रूप में देता है। हर topic में: - कब इस्तेमाल करें - Core pattern - Python template

---

1. Big O Notation (Complexity Analysis)

Use when: Algorithm की efficiency समझनी हो

Template

```python
# Time Complexity: O(?)
# Space Complexity: O(?)

def algo(n):
    for i in range(n):      # O(n)
        pass
```

Common Patterns

- O(1) → Constant
- O(n) → Single loop
- O(log n) → Binary search
- O(n log n) → Merge sort
- O(n^2) → Nested loops

---

2. Array

Use when: Index-based access, contiguous data

Patterns

- Two pointers
- Prefix sum
- In-place update

```python
# Two pointers
l, r = 0, len(arr) - 1
while l < r:
    l += 1
    r -= 1
```

```
# Prefix sum
prefix = [0]
for x in arr:
    prefix.append(prefix[-1] + x)
```

---

3. List (Dynamic Array)

Use when: Resizable array needed

```
lst = []
lst.append(x)      # O(1) amortized
lst.pop()          # O(1)
lst.insert(i, x)   # O(n)
```

Pattern: **Read–Write pointer**

---

4. Dictionary (Hash Map)

Use when: Fast lookup, frequency counting

```
freq = {}
for x in arr:
    freq[x] = freq.get(x, 0) + 1
```

Patterns: - Frequency map - Two Sum - Group By

---

5. Tuple

Use when: Immutable & hashable data

```
point = (x, y)
visited = set()
visited.add(point)
```

---

6. String

Use when: Text, substring problems

Patterns

- Palindrome
- Sliding window
- Frequency count

```
# Palindrome
s == s[::-1]

# Build string efficiently
res = []
res.append(ch)
ans = ''.join(res)
```

---

7. Math

Use when: Number-based logic

```
# GCD
while b:
    a, b = b, a % b

# Prime check
for i in range(2, int(n**0.5)+1):
    pass
```

Patterns: - Digit extraction - Bit manipulation

---

8. Recursion

Use when: Problem can be broken into smaller subproblems

Template

```
def solve(n):
    if n == 0:         # base case
        return
    solve(n-1)        # recursive call
```

Memoization

```
from functools import lru_cache

@lru_cache(None)
```

```python
def dp(n):
    pass
```

---

9. Linked List

Patterns

- Traversal
- Fast & Slow pointer
- Reverse

```python
prev = None
cur = head
while cur:
    nxt = cur.next
    cur.next = prev
    prev = cur
    cur = nxt
```

---

10. Stack

Use when: LIFO, previous elements matter

Patterns: - Valid parentheses - Monotonic stack

```python
stack = []
stack.append(x)
stack.pop()
```

---

11. Queue

Use when: FIFO, BFS

```python
from collections import deque
q = deque()
q.append(x)
q.popleft()
```

Pattern: **Breadth First Search (BFS)**

---

12. Hash Set / Hash Table

Use when: Uniqueness, fast membership

```
seen = set()
if x in seen:
    pass
```

Patterns: - Deduplication - Caching

---

13. Searching

Binary Search Template

```
l, r = 0, len(arr)-1
while l <= r:
    mid = (l+r)//2
```

Use when data is **sorted**

---

14. Sorting

Patterns: - Merge Sort (Divide & Conquer) - Quick Sort

```
arr.sort()
sorted(arr, key=lambda x: x)
```

---

15. Sliding Window ⭐

Most Important Pattern

Fixed Window

```
window = sum(arr[:k])
for i in range(k, n):
    window += arr[i] - arr[i-k]
```

Variable Window

```
l = 0
for r in range(n):
    while condition:
        l += 1
```

Master Problem-Solving Template

```
def solve():
    # 1. Edge cases
    # 2. Choose DS
    # 3. Apply pattern
    # 4. Optimize
    pass
```

🦙Must-Remember Patterns

- Two Pointers
- Sliding Window
- Binary Search
- Fast & Slow Pointer
- BFS / DFS
- Dynamic Programming
- Monotonic Stack

✅**If you want:** - Topic-wise practice problems - LeetCode pattern mapping - Interview cheat sheet (1-page) - Hindi explanation per topic

Just tell me ♀️