Overview

Prefix Sum Cookbook — Arrays, Linked Lists, 2D (Light Theme)
Generated: 2025-10-14 05:01:47

This guide explains prefix sums (running sums) with concise, production-ready Python examples.
Includes:

- 1D arrays (range queries)
- Singly linked lists (in-place and new-list variants)
- 2D matrices (submatrix queries)

Array Prefix Sum

```
1) Array Prefix Sum
-------------------
Definition: ps[i] = nums[0] + nums[1] + ... + nums[i]

Python:
    def prefix_sum(nums):
        ps = [0]*len(nums)
        running = 0
        for i, x in enumerate(nums):
            running += x
            ps[i] = running
        return ps

    def range_sum(ps, l, r):
        # inclusive indices
        return ps[r] - (ps[l-1] if l > 0 else 0)

Example:
    nums = [3, -1, 4, 2]
    ps = prefix_sum(nums)            # [3, 2, 6, 8]
    assert range_sum(ps, 1, 3) == 5  # -1 + 4 + 2
```

Linked List Prefix Sum

2) Linked List Prefix Sum

ListNode:

```
    class ListNode:
        def __init__(self, val=0, nxt=None):
            self.val, self.next = val, nxt
```

(a) In-place (mutates values to running sum):

```
    def prefix_sum_inplace(head):
        running = 0
        cur = head
        while cur:
            running += cur.val
            cur.val = running
            cur = cur.next
        return head
```

(b) New list (does not mutate input):

```
    def prefix_sum_newlist(head):
        dummy = ListNode(0)
        tail = dummy
        running = 0
        cur = head
        while cur:
            running += cur.val
            tail.next = ListNode(running)
            tail = tail.next
            cur = cur.next
        return dummy.next
```

Notes:

- Two-pointer slow/fast pattern is for middle detection, not needed for prefix sums.
- Keep it single-pass; O(n) time, O(1) extra for in-place, O(n) for new list variant.

2D Prefix Sum

```
3) 2D Prefix Sum (Integral Image)
---------------------------------
Goal: O(1) submatrix sum after O(m*n) precompute.

Build:
    def prefix_sum_2d(mat):
        if not mat or not mat[0]: return [[0]]
        m, n = len(mat), len(mat[0])
        ps = [[0]*(n+1) for _ in range(m+1)]
        for i in range(1, m+1):
            row_sum = 0
            for j in range(1, n+1):
                row_sum += mat[i-1][j-1]
                ps[i][j] = ps[i-1][j] + row_sum
        return ps

Query (inclusive r1..r2, c1..c2):
    def sum_region(ps, r1, c1, r2, c2):
        r1+=1; c1+=1; r2+=1; c2+=1
        return ps[r2][c2] - ps[r1-1][c2] - ps[r2][c1-1] + ps[r1-1][c1-1]

Example:
    mat = [
        [1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]
    ]
    ps = prefix_sum_2d(mat)
    assert sum_region(ps, 0, 0, 1, 1) == 12   # 1+2+4+5
```

Tips

Tips & Pitfalls

- Use prefix sums for O(1) range queries with O(n) preprocessing.
- For immutable arrays, store ps; for frequent updates, consider Fenwick Tree / Segment Tree.
- Keep partition/pruning rules in SQL analogies: avoid wrapping columns in functions when indexing matters.
- Linked list variant is handy when you cannot index randomly but can mutate as you scan.