Prefix Sum Patterns — Five Practical Examples

This pack demonstrates five classic prefix-sum variants with clean Python code: 1) **subarray_sum_k**: count subarrays with sum == k 2) **NumMatrix**: O(1) 2D range sums 3) **apply_range_adds**: batch range increments via difference array 4) **subarray_sum_divisible_by_k**: count subarrays where sum % K == 0 5) **NumArray**: immutable 1D range sums

subarray_sum_k (exact sum)

```
pref = 0
    seen = defaultdict(int)
    seen[0] = 1  # empty prefix
    ans = 0
    for x in nums:
        pref += x
        ans += seen[pref - k]
        seen[pref] += 1
    return ans

def main():
    nums = [1, 1, 1]
    print("Array:", nums, "k=2 ->", subarray_sum_k(nums, 2))  # 2

if __name__ == "__main__":
    main()
```

NumMatrix (2D prefix)

```
from typing import List

class NumMatrix:
    def __init__(self, M: List[List[int]]):
        m = len(M)
        n = len(M[0]) if m else 0
        self.P = [[0] * (n + 1) for _ in range(m + 1)]
        for i in range(1, m + 1):
            for j in range(1, n + 1):
                self.P[i][j] = (
                    M[i - 1][j - 1]
                    + self.P[i - 1][j]
                    + self.P[i][j - 1]
                    - self.P[i - 1][j - 1]
                )

    def sumRegion(self, r1: int, c1: int, r2: int, c2: int) -> int:
        P = self.P
        return P[r2 + 1][c2 + 1] - P[r1][c2 + 1] - P[r2 + 1][c1] + P[r1][c1]

def main():
    M = [
        [3, 0, 1, 4, 2],
        [5, 6, 3, 2, 1],
        [1, 2, 0, 1, 5],
        [4, 1, 0, 1, 7],
        [1, 0, 3, 0, 5],
```

```
    ]
    nm = NumMatrix(M)
    print("Sum (2,1)-(4,3):", nm.sumRegion(2,1,4,3))  # 8
    print("Sum (1,1)-(2,2):", nm.sumRegion(1,1,2,2))  # 11

if __name__ == "__main__":
    main()
```

apply_range_adds (diff array)

```
from typing import List, Tuple

def apply_range_adds(n: int, updates: List[Tuple[int, int, int]]) -> List[int]:
    diff = [0] * (n + 1)
    for l, r, v in updates:
        diff[l] += v
        if r + 1 < len(diff):
            diff[r + 1] -= v
    cur = 0
    out = [0] * n
    for i in range(n):
        cur += diff[i]
        out[i] = cur
    return out

def main():
    n = 5
    updates = [(1, 3, 2), (2, 4, 3)]
    print("n=5, updates=", updates)
    print("Result:", apply_range_adds(n, updates))  # [0,2,5,5,3]

if __name__ == "__main__":
    main()
```

subarray_sum_divisible_by_k (mod)

```
from collections import defaultdict
from typing import List

def subarray_sum_divisible_by_k(nums: List[int], K: int) -> int:
    cnt = defaultdict(int)
    cnt[0] = 1     # empty prefix remainder
    pref = 0
    ans = 0
    for x in nums:
        pref += x
        r = pref % K
        ans += cnt[r]   # all earlier prefixes with same remainder
        cnt[r] += 1
    return ans

def main():
    nums = [4,5,0,-2,-3,1]
    K = 5
    print("Array:", nums, "K=", K, "->", subarray_sum_divisible_by_k(nums, K))  # 7

if __name__ == "__main__":
```

```
    main()
```

NumArray1D (immutable range sum)

```
from typing import List

class NumArray:
    def __init__(self, nums: List[int]):
        self.P = [0]
        for x in nums:
            self.P.append(self.P[-1] + x)

    def sumRange(self, i: int, j: int) -> int:
        return self.P[j + 1] - self.P[i]

def main():
    arr = NumArray([-2, 0, 3, -5, 2, -1])
    print("sumRange(0,2)=", arr.sumRange(0,2))  # 1
    print("sumRange(2,5)=", arr.sumRange(2,5))  # -1
    print("sumRange(0,5)=", arr.sumRange(0,5))  # -3

if __name__ == "__main__":
    main()
```

References: algomaster patterns & video walkthroughs (see problem comments).