# Sliding Window — Coding Interview Notes (Light Theme)

## General Pattern Template

```
def fn(arr):
    left = ans = curr = 0

    for right in range(len(arr)):
        # do logic here to add arr[right] to curr

        while WINDOW_CONDITION_BROKEN:
            # remove arr[left] from curr
            left += 1

        # update ans

    return ans
```

**Concept:**
The **Sliding Window** pattern keeps a moving window over a sequence and updates state as the window expands and (optionally) shrinks. It achieves linear scans where naive solutions would re-scan or use nested loops.

**When to use:** contiguous subarrays/substrings, maxima/minima over ranges, constraints like distinct count or sum thresholds.
**Complexity:** Typically O(n) time and O(1)–O(k) extra space depending on the state tracked.

## Key Ideas

1. The window is defined by indices [left, right].
2. Expand right each step; shrink left while a constraint is violated.
3. Maintain rolling state (sum, counts, freq map) incrementally.
4. Two flavors: fixed-size windows (exact length k) and variable-size windows (bounded by a condition).

## Example 1: Max Sum of Subarray of Size k (Fixed Window)

**Goal:** Given an array of integers and integer k, find the maximum sum of any contiguous subarray of size k.
**Approach:** Grow window to size k, then for each step add arr[right] and remove arr[left] to keep size k.

```
def max_sum_subarray_k(nums, k):
    left = 0
    curr = 0
    ans = float('-inf')

    for right in range(len(nums)):
        curr += nums[right]

        if right - left + 1 == k:
            ans = max(ans, curr)
            curr -= nums[left]
            left += 1

    return ans if ans != float('-inf') else 0
```

## Example 2: Longest Substring Without Repeating Characters (Variable Window)

**Goal:** Given a string s, return the length of the longest substring without repeating characters.
**Approach:** Use a frequency/index map. Expand right; while duplicate seen in window, move left to shrink.

```
def length_of_longest_substring(s):
    last = {}   # char -> latest index
    left = 0
    ans = 0

    for right, ch in enumerate(s):
        if ch in last and last[ch] >= left:
            left = last[ch] + 1  # shrink to exclude previous ch
        last[ch] = right
        ans = max(ans, right - left + 1)

    return ans
```

## Example 3: Smallest Subarray with Sum ≥ Target (Positive Integers)

**Goal:** Given an array of *positive* integers and target, find the minimal length of a contiguous subarray of which the sum ≥ target. Return 0 if none.
**Approach:** Expand right adding to sum; while sum ≥ target, update answer and shrink left.

```
def min_subarray_len(target, nums):
    left = 0
    curr = 0
    ans = float('inf')

    for right in range(len(nums)):
        curr += nums[right]

        while curr >= target:
            ans = min(ans, right - left + 1)
            curr -= nums[left]
            left += 1
    return 0 if ans == float('inf') else ans
```

## Summary Table

| Problem Type | Constraint | State Tracked | Example |
|---|---|---|---|
| Fixed-size window | Window length = k | Running sum | Max sum of subarray of size k |
| Variable-size window | No repeats | Last index / freq map | Longest substring without repeats |
| Variable-size window | Sum ≥ target (positives) | Running sum | Smallest subarray ≥ target |