# Python 3 Cheatsheet: Key Algorithms

## Binary Search

Template for `left < right`:

```
def search(nums, target):
    l, r = 0, len(nums)
    while l < r:
        mid = l + (r - l) // 2
        if nums[mid] >= target: # Condition
            r = mid
        else:
            l = mid + 1
    return l
```

## Sliding Window

Base template.

```
def slidingWindow(s):
    left = 0
    res = 0
    curr = 0
    for right in range(len(s)):
        # Add s[right] to window
        curr += 1

        while not valid(curr): # Shrink condition
            # Remove s[left]
            curr -= 1
            left += 1

        res = max(res, right - left + 1)
    return res
```

## Heap (Priority Queue)

Min-heap by default. Negate values for Max-heap.

```
import heapq
arr = [3, 1, 4]
heapq.heapify(arr)        # O(N) -> [1, 3, 4]
heapq.heappush(arr, 2)    # O(logN)
small = heapq.heappop(arr)# O(logN) -> 1
```

```
# K Largest/Smallest
heapq.nlargest(3, nums)
heapq.nsmallest(3, nums, key=lambda x: x[1])

# Merge Sorted
list(heapq.merge([1,5], [2,4])) # [1, 2, 4, 5]
```

## Matrix Traversal

```
rows, cols = len(grid), len(grid[0])
directions = [(0,1), (0,-1), (1,0), (-1,0)]
for dr, dc in directions:
    nr, nc = r + dr, c + dc
    if 0 <= nr < rows and 0 <= nc < cols:
        # Valid neighbor
```

## Backtracking (Permutations)

```
def backtrack(path, options):
    if goal(path):
        res.append(path[:])
        return
    for i in range(len(options)):
        path.append(options[i])
        backtrack(path, options[:i] + options[i+1:])
        path.pop()
```