Python Arrays Cheat Sheet: Lists, Arrays, and DSA Patterns

1. Array Basics in Python

Python does not have a primitive array type; the most common dynamic array is the **list**. For numerical work, `numpy.ndarray` is often used. For fixed-type arrays, there's the built-in `array.array`.

A. Native Python List

```
# Creation
arr = [1, 2, 3, 4]

# Indexing & Slicing
first = arr[^0]; last = arr[-1]
slice = arr[1:3]       # [2, 3]

# Modifying
arr.append(5)           # [1,2,3,4,5]
arr.insert(1, 8)        # [1,8,2,3,4,5]
arr.remove(3)           # [1,8,2,4,5]
del arr[^0]              # [8,2,4,5]
arr.pop()               # removes last element; arr now [8,2,4]

# Combining & Copying
arr2 = arr.copy()
arr.extend([6,7])       # [8,2,4,6,7]
joined = arr + [10, 20]
```

B. array.array (Efficient, type-coded, numeric only)

```
from array import array
a = array('i', [1,2,3,4])    # 'i' for int32
a[^1] = 99
for num in a: print(num)
```

C. numpy.ndarray (High-performance numerical arrays, see NumPy cheat sheet)

2. Array Patterns for DSA (with Lists)

A. Traversal Patterns

- **Simple loop**

```
for v in arr:
    print(v)
```

- **By index**

```
for i in range(len(arr)):
    print(arr[i])
```

- **Reverse**

```
for v in reversed(arr): ...
# or
arr[::-1]
```

B. Searching

- **Linear search**

```
def find(arr, x):
    for i, v in enumerate(arr):
        if v == x:
            return i
    return -1
```

- **Binary search** (sorted arrays)

```
def binary_search(arr, target):
    l, r = 0, len(arr)-1
    while l <= r:
        m = (l+r)//2
        if arr[m] == target: return m
        elif arr[m] < target: l = m+1
        else: r = m-1
    return -1
```

C. Modifying Arrays In-place

- **Move all zeros to the end**

```
def move_zeros(arr):
    pos = 0
    for val in arr:
        if val != 0:
            arr[pos] = val
            pos += 1
    for i in range(pos, len(arr)):
        arr[i] = 0
```

- **Reverse in-place**

```
def reverse(arr):
    l, r = 0, len(arr)-1
    while l < r:
        arr[l], arr[r] = arr[r], arr[l]
        l += 1; r -= 1
```

D. Sliding Window Patterns

- **Fixed window sum**

```
def max_sum_subarray(arr, k):
    curr = max_sum = sum(arr[:k])
    for i in range(k, len(arr)):
        curr += arr[i] - arr[i-k]
        max_sum = max(max_sum, curr)
    return max_sum
```

- **Variable window (for substring/array problems)**

```
def min_len_subarray(arr, target):
    n = len(arr)
    s = 0; summ = 0; res = n+1
    for e in range(n):
        summ += arr[e]
        while summ >= target:
            res = min(res, e-s+1)
            summ -= arr[s]
            s += 1
    return res if res <= n else -1
```

E. Two Pointers

- **Remove duplicates from sorted array (in place)**

```
def remove_duplicates(arr):
    if not arr: return 0
    idx = 0
    for i in range(1, len(arr)):
        if arr[i] != arr[idx]:
            idx += 1
            arr[idx] = arr[i]
    return idx + 1
```

- **Partition array (Dutch National Flag)**

```
def partition(arr, pivot):
    l, r = 0, len(arr)-1
    while l <= r:
        while l < len(arr) and arr[l] < pivot: l +=1
        while r >=0 and arr[r] >= pivot: r -= 1
        if l < r:
            arr[l], arr[r] = arr[r], arr[l]
```

F. Subarrays/Subsets

- **All subarrays**

```
for i in range(len(arr)):
    for j in range(i, len(arr)):
        print(arr[i:j+1])
```

- **All subsets (Power set)**

```
def subsets(nums):
    res = [[]]
    for n in nums:
        res += [sub + [n] for sub in res]
    return res
```

G. Prefix Sum

- **Sum of elements in range O(1) after pre-processing**

```
ps = [^0] * (len(arr) +1)
for i in range(len(arr)):
    ps[i+1] = ps[i] + arr[i]
# Sum arr[i:j] = ps[j] - ps[i]
```

H. Frequency Counter

- **Count elements**

```
from collections import Counter
counts = Counter(arr)
most_common = counts.most_common(1)
```

I. Sorting Patterns

- **Built-in**

```
arr.sort()        # In-place
sorted_arr = sorted(arr)    # Returns new
arr.sort(reverse=True)
```

- **Custom key**

```
arr.sort(key=lambda x: x[^1])
```

3. Multi-dimensional Arrays

- **2D list**

```
grid = [[^0]*4 for _ in range(3)]  # 3 rows, 4 columns
grid[^1][^2] = 7
```

- **Matrix traversal**

```
for row in grid:
    for val in row:
        print(val)
```

- **Transpose**

```
transposed = list(zip(*grid))
```

- **Row/col sum**

```
row_sum = [sum(row) for row in grid]
col_sum = [sum(col) for col in zip(*grid)]
```

4. Patterns for Coding Interviews

- **Kadane's Algorithm (max subarray sum)**

```
def max_subarray(arr):
    max_end = max_so_far = arr[^0]
    for x in arr[1:]:
        max_end = max(x, max_end + x)
        max_so_far = max(max_so_far, max_end)
    return max_so_far
```

- **Trapping Rain Water**

```
def trap(height):
    left, right = 0, len(height) - 1
    left_max = right_max = water = 0
    while left < right:
        if height[left] < height[right]:
            left_max = max(left_max, height[left])
            water += left_max - height[left]
            left += 1
        else:
            right_max = max(right_max, height[right])
            water += right_max - height[right]
            right -= 1
    return water
```

- **Next Greater Element**

```
def next_greater(arr):
    stack, res = [], [-1]*len(arr)
    for i in range(len(arr)):
        while stack and arr[i] > arr[stack[-1]]:
            res[stack.pop()] = arr[i]
        stack.append(i)
    return res
```

- **Rotate Array k Steps (cyclic, in place)**

```
def rotate(arr, k):
    n = len(arr)
    k %= n
    arr[:] = arr[-k:] + arr[:-k]
```

5. Pro Tips

- **Copy list:** `b = arr.copy()` or `b = arr[:]` (not `b = arr`, which aliases)
- **List comprehensions:** `[x**2 for x in arr if x>0]`
- **Deep copy (nested arrays):** `import copy; copy.deepcopy(arr)`
- **Flatten nested lists:** `flat = [item for sub in arr for item in sub]`

6. Recommended Array Libraries

- **NumPy** for large, numerical arrays and matrices.
- **array.array** for memory-efficient fixed-type arrays.
- **collections.deque** for fast append/pop from both ends (useful for sliding window).

7. Array Resources

- Official docs: [docs.python.org list, array][1]
- LeetCode/GeeksforGeeks “Array” sections: for hundreds of array DSA problems and patterns.
- NumPy cheat sheet: for vectorized numerical data.