Python Interview Questions Cheat Sheet

--- NUMBERS ---

1. Reverse an integer without converting to string
   Hint: Use modulo % 10 and integer division //
   Code:

```
def reverse_int(n):
    rev = 0
    while n > 0:
        rev = rev * 10 + n % 10
        n //= 10
    return rev
```

2. Check if a number is prime efficiently
   Hint: Check divisibility up to sqrt(n)
   Code:

```
import math
def is_prime(n):
    if n <= 1: return False
    for i in range(2, int(math.sqrt(n)) + 1):
        if n % i == 0:
            return False
    return True
```

3. Find GCD and LCM without math module
   Hint: Use Euclidean algorithm
   Code:

```
def gcd(a,b):
    while b: a,b = b, a%b
    return a
def lcm(a,b):
    return a*b//gcd(a,b)
```

4. Sum large list of floats accurately
   Hint: Use math.fsum to avoid precision issues
   Code:

```
import math
math.fsum([0.1]*10)
```

5. Modular exponentiation efficiently
   Hint: Use exponentiation by squaring
   Code:

```
def mod_pow(a,b,m):
    result = 1
    a = a % m
    while b > 0:
        if b % 2 == 1:
            result = (result * a) % m
        a = (a*a) % m
        b //= 2
    return result
```

6. Detect perfect squares
   Hint: Compare int(n**0.5)**2 with n
   Code:
   ```
   def is_square(n):
       return int(n**0.5)**2 == n
   ```

7. Count digits in an integer
   Hint: Repeated division by 10
   Code:
   ```
   def count_digits(n):
       count = 0
       while n:
           n //= 10
           count += 1
       return count
   ```

8. Generate all divisors of a number
   Hint: Loop up to sqrt(n)
   Code:
   ```
   def divisors(n):
       divs = set()
       for i in range(1,int(n**0.5)+1):
           if n % i == 0:
               divs.add(i)
               divs.add(n//i)
       return divs
   ```

9. Integer to Roman numeral
   Hint: Subtract values iteratively
   Code:
   ```
   def int_to_roman(num):
       val = [1000,900,500,400,100,90,50,40,10,9,5,4,1]
       sym = ["M","CM","D","CD","C","XC","L","XL","X","IX","V","IV","I"]
       res = ""
       for i,v in enumerate(val):
           while num >= v:
               num -= v
               res += sym[i]
       return res
   ```

10. Roman numeral to integer
    Hint: Parse left to right, handle subtraction
    Code:
    ```
    def roman_to_int(s):
        rom = {'I':1,'V':5,'X':10,'L':50,'C':100,'D':500,'M':1000}
        total = 0
        prev = 0
        for ch in reversed(s):
            curr = rom[ch]
            if curr < prev: total -= curr
            else: total += curr
            prev = curr
        return total
    ```

11. Count number of 1s in binary representation
    Hint: Use n & (n-1) trick
    Code:
    def count_ones(n):
        count = 0
        while n:
            n &= n-1
            count += 1
        return count

12. Find missing number in 1..n
    Hint: Sum formula or XOR
    Code:
    def missing_number(arr, n):
        return n*(n+1)//2 - sum(arr)

13. Find duplicate in array
    Hint: Use set or XOR
    Code:
    def find_duplicate(arr):
        seen = set()
        for x in arr:
            if x in seen: return x
            seen.add(x)
        return None

14. Power of two check
    Hint: n>0 and n&(n-1)==0
    Code:
    def is_power_of_two(n):
        return n>0 and (n&(n-1))==0

15. Factorial (iterative/recursive)
    Code:
    def factorial(n):
        if n==0: return 1
        return n*factorial(n-1)

--- STRINGS ---
16. Reverse a string
    Hint: s[::-1]

17. Check if string is palindrome
    Hint: s == s[::-1]

18. Count vowels/consonants
    Code:
    def count_vowels_consonants(s):
        vowels = set('aeiouAEIOU')
        v = c = 0
        for ch in s:
            if ch.isalpha():

```
            if ch in vowels: v+=1
            else: c+=1
    return v,c
```

19. First non-repeating character
    Code:
    ```
    from collections import Counter
    def first_unique(s):
        c = Counter(s)
        for ch in s:
            if c[ch]==1: return ch
        return None
    ```

20. Group anagrams
    Code:
    ```
    from collections import defaultdict
    def group_anagrams(words):
        d = defaultdict(list)
        for w in words:
            d[tuple(sorted(w))].append(w)
        return list(d.values())
    ```

21. Longest substring without repeating chars
    Hint: Sliding window + set

22. Longest palindromic substring
    Hint: Expand around center

23. String compression
    Code:
    ```
    def compress(s):
        res = ""
        i=0
        while i<len(s):
            count=1
            while i+1<len(s) and s[i]==s[i+1]:
                count+=1
                i+=1
            res+=s[i]+str(count)
            i+=1
        return res
    ```

24. Character frequency
    Code:
    ```
    from collections import Counter
    Counter(s)
    ```

25. Substring search without `in`
    Hint: Implement naive or KMP

26. Check rotation of string
    Hint: Concatenate and check substring

27. Count substrings/pattern occurrences
    Hint: Sliding window or KMP

28. Convert string to int safely
    Hint: Handle exceptions or use int()

29. Remove duplicates from string
    Hint: Use set or OrderedDict

30. Longest common prefix
    Hint: Compare character by character

--- LISTS ---
31. Largest/smallest element
    Code: max(lst), min(lst)

32. Remove duplicates
    Code: list(set(lst))

33. Rotate list by k positions
    Code: lst[-k:]+lst[:-k]

34. Merge two sorted lists
    Hint: Two pointers

35. Count occurrences
    Code: from collections import Counter; Counter(lst)

36. Sliding window max sum
    Hint: Prefix sum or deque

37. Kadane's algorithm
    Code:
    ```
    def max_subarray(lst):
        max_ending=max_so_far=lst[0]
        for x in lst[1:]:
            max_ending = max(x, max_ending+x)
            max_so_far = max(max_so_far, max_ending)
        return max_so_far
    ```

38. Find duplicates in list
    Hint: Use Counter or set

39. Intersection of two lists
    Hint: Use set intersection

40. Subarray with given sum
    Hint: Prefix sum or sliding window

41. Two sum problem
    Hint: Use hashmap/dictionary

42. Move zeros to end

```
    Code:
    def move_zeros(arr):
        res = [x for x in arr if x!=0]
        res += [0]*arr.count(0)
        return res

43. Rotate matrix 90 degrees
    Hint: Transpose + reverse rows

44. Spiral matrix
    Hint: Layer by layer

45. Merge intervals
    Hint: Sort + merge overlapping

--- TUPLES ---
46. Swap values without temp
    Code: a,b = b,a

47. Tuple unpacking
    Code: x,y,*rest = (1,2,3,4)

48. Count occurrences in tuple
    Code: t.count(val)

49. Convert list to tuple
    Code: tuple(lst)

50. Immutable operations
    Hint: Tuples are immutable, copy or convert to list

--- SETS ---
51. Union/intersection/difference
    Code: set1 | set2, set1 & set2, set1 - set2

52. Subset/superset check
    Code: set1 <= set2

53. Remove duplicates from list
    Code: list(set(lst))

54. Symmetric difference
    Code: set1 ^ set2

55. Check unique elements
    Code: len(lst) == len(set(lst))

--- DICTIONARIES ---
56. Count word frequency
    Code: from collections import Counter; Counter(words)

57. Merge dictionaries
    Code: {**d1, **d2}
```

58. Invert dictionary
    Code: {v:k for k,v in d.items()}

59. Sort dict by value
    Code: sorted(d.items(), key=lambda x:x[1])

60. Default dict usage
    Code:
    from collections import defaultdict
    d = defaultdict(list)

61. Nested dictionary access
    Hint: Use get() safely

62. Find key with max value
    Code: max(d, key=d.get)

63. Remove key safely
    Code: d.pop('key', None)

64. Check if dict is empty
    Code: not d

65. Count occurrences from list
    Code: d = Counter(lst)

--- COMBINED / ADVANCED ---
66. Flatten nested list
    Hint: Use recursion or itertools

67. Flatten nested dict
    Hint: Recursion

68. Merge k sorted lists
    Hint: Heapq merge

69. LRU Cache implementation
    Hint: Use OrderedDict or functools.lru_cache

70. Detect cycle in linked list
    Hint: Floyd's Tortoise and Hare

71. Serialize/deserialize nested structures
    Hint: Use json module or pickle

72. Find median of sorted arrays
    Hint: Binary search

73. Sliding window max/min
    Hint: Use deque

74. Top k frequent elements

    Hint: Use heapq

75. Count islands in grid
    Hint: DFS/BFS

76. Word ladder / transformation
    Hint: BFS

77. Valid parentheses
    Hint: Stack

78. Minimum spanning tree
    Hint: Kruskal/Prim

79. Dijkstra shortest path
    Hint: PriorityQueue

80. N-Queens
    Hint: Backtracking

81. Sudoku solver
    Hint: Backtracking

82. Combination sum
    Hint: Backtracking

83. Permutations / combinations
    Hint: itertools or recursion

84. Reverse linked list
    Hint: Iterative/recursive

85. Detect loop in linked list
    Hint: Fast/slow pointers

86. Merge two binary trees
    Hint: DFS/BFS

87. Binary tree traversal
    Hint: Inorder, preorder, postorder, level-order

88. Lowest common ancestor
    Hint: Recursive DFS

89. Validate BST
    Hint: Inorder traversal

90. Serialize/deserialize BST
    Hint: Preorder + Null markers

91. Heap operations
    Hint: Use heapq module

92. Sliding window substring problems
    Hint: Hashmap + window

93. Subarray sum equals k
    Hint: Prefix sum + hashmap

94. Longest consecutive sequence
    Hint: Set for O(n) solution

95. Number of connected components
    Hint: DFS/BFS

96. Topological sort
    Hint: DFS or Kahn's algorithm

97. Detect cycle in directed graph
    Hint: DFS with recursion stack

98. Minimum window substring
    Hint: Sliding window + hashmap

99. Regular expression matching
    Hint: DP / regex module

100. Implement stack with min()
    Hint: Maintain auxiliary stack

101. Implement queue with 2 stacks
    Hint: Push-pop strategy

102. Evaluate reverse Polish notation
    Hint: Stack

103. LRU cache using dict + linked list
    Hint: OrderedDict

104. Design hash map
    Hint: Buckets + linked list

105. Serialize/deserialize nested structures
    Hint: Recursion or json module