DevRev CodeSignal Coding Prep: Solutions for All Questions

Hey Somanath! You've got a solid foundation in Python, AI/ML, and APIs from your work at T-Machine and Soul AI. This guide provides simple Python solutions for all 34 coding questions you listed, perfect for DevRev's first-round CodeSignal test. Each solution is easy to understand, with a clear explanation of how it works. Practice these to ace the interview!

Preparation Tips

- **Day 1**: Solve 17 questions (Arrays, Strings, Linked Lists, 4-5 hours).
- **Day 2**: Solve 17 questions (Trees, Stacks, Graphs, DP, 4-5 hours) and practice on CodeSignal's "Arcade" section.
- **CodeSignal Tips**: Explain your logic aloud, test edge cases, and aim for 30-40 minutes per problem.

Arrays & Hashing

1. Two Sum

Problem: Find indices of two numbers in an array that add up to a target.
Example: Input: nums = [2, 7, 11, 15], target = 9, Output: [0, 1]

```
def twoSum(nums, target):
    seen = {}
    for i, num in enumerate(nums):
        complement = target - num
        if complement in seen:
            return [seen[complement], i]
        seen[num] = i
    return []
```

Explanation:

- We need two numbers that add to `target`.
- Use a dictionary (`seen`) to store each number and its index.
- For each number, calculate `complement = target - num`. If `complement` is in `seen`, we found the pair.
- If not, add the number and index to `seen`.
- **Edge Cases**: No solution, duplicate numbers.

---

2. Subarray Sum Equals K

Problem: Find the number of subarrays with sum equal to k.
Example: Input: nums = [1, 1, 1], k = 2, Output: 2

```
def subarraySum(nums, k):
    count = 0
    curr_sum = 0
    prefix_sums = {0: 1}
    for num in nums:
        curr_sum += num
        if curr_sum - k in prefix_sums:
            count += prefix_sums[curr_sum - k]
        prefix_sums[curr_sum] = prefix_sums.get(curr_sum, 0) + 1
    return count
```

Explanation:

- Track cumulative sum (`curr_sum`) and count subarrays summing to `k`.
- Use a dictionary to store how many times each sum occurs.
- For each number, add to `curr_sum`. If `curr_sum - k` exists, it means a subarray sums to `k`.
- Update `prefix_sums` with the current sum.
- **Edge Cases**: Empty array, k = 0.

---

3. Product of Array Except Self

Problem: Return an array where each element is the product of all other elements.
Example: Input: nums = [1, 2, 3, 4], Output: [24, 12, 8, 6]

```
def productExceptSelf(nums):
    n = len(nums)
    output = [1] * n
    left = 1
    for i in range(n):
        output[i] = left
        left *= nums[i]
    right = 1
    for i in range(n-1, -1, -1):
        output[i] *= right
        right *= nums[i]
    return output
```

Explanation:

- For each index, calculate product of all numbers except that index.
- First pass: Multiply left-side products into `output`.
- Second pass: Multiply right-side products into `output`.
- Avoid division to handle zeros.
- **Edge Cases**: Single element, zeros in array.

---

4. Longest Consecutive Sequence

Problem: Find the length of the longest consecutive sequence.
Example: Input: nums = [100, 4, 200, 1, 3, 2], Output: 4 ([1, 2, 3, 4])

```
def longestConsecutive(nums):
    num_set = set(nums)
    longest = 0
    for num in nums:
        if num - 1 not in num_set:
            current = num
            streak = 1
            while current + 1 in num_set:
                current += 1
                streak += 1
            longest = max(longest, streak)
    return longest
```

Explanation:

- Convert array to set for fast lookup.
- For each number, if it's the start of a sequence (no `num-1`), count consecutive numbers.
- Track longest sequence found.
- **Edge Cases**: Empty array, duplicates.

---

5. Majority Element

Problem: Find the element that appears more than n/2 times.
Example: Input: nums = [2, 2, 1, 1, 1, 2, 2], Output: 2

```
def majorityElement(self, nums):
    count = 0
    candidate = None
    for num in nums:
        if count == 0:
            candidate = num
        count += (1 if num == candidate else -1)
    return candidate
```

Explanation:

- Use Boyer-Moore Voting Algorithm.
- Pick a `candidate` and count votes. If same number, increase count; else, decrease.
- When count is 0, pick new candidate.
- Majority element always wins.
- **Edge Cases**: Single element, guaranteed majority.

Strings

6. Valid Anagram

Problem: Check if two strings are anagrams.
Example: Input: s = "anagram", t = "nagaram", Output: true

```
def isAnagram(s, t):
    if len(s) != len(t):
        return False
    count = {}
    for char in s:
        count[char] = count.get(char, 0) + 1
    for char in t:
        count[char] = count.get(char, 0) - 1
        if count[char] == 0:
            del count[char]
    return len(count) == 0
```

Explanation:

- Anagrams have same letters with same counts.
- Use dictionary to count characters in `s`, then decrease for `t`.
- If dictionary is empty, strings are anagrams.
- **Edge Cases**: Different lengths, empty strings.

---

7. Longest Substring Without Repeating Characters

Problem: Find length of longest substring with no repeating characters.
Example: Input: s = "abcabcbb", Output: 3 ("abc")

```
def lengthOfLongestSubstring(s):
    seen = {}
    max_length = 0
    start = 0
    for end, char in enumerate(s):
        if char in seen and seen[char] >= start:
            start = seen[char] + 1
        else:
            max_length = max(max_length, end - start + 1)
        seen[char] = end
    return max_length
```

**■

Explanation:

- Use sliding window with a dictionary to track character positions.
- Move `start` when a repeat is found, update `max_length`.
- Store current position of each character.
- **Edge Cases**: Empty string, single character.

---

8. Palindrome Check

Problem: Check if a string is a palindrome.
Example: Input: s = "racecar", Output: true

```
def isPalindrome(s):
    s = ''.join(c.lower() for c in s if c.isalnum())
    left, right = 0, len(s) - 1
    while left < right:
        if s[left] != s[right]:
            return False
        left += 1
        right -= 1
    return True
```

Explanation:

- Clean string: lowercase, remove non-alphanumeric characters.
- Use two pointers (`left` , `right`) to compare characters from ends.
- If mismatch, not a palindrome.
- **Edge Cases**: Empty string, single character.

---

9. Group Anagrams

Problem: Group anagrams together from a list of strings.
Example: Input: ["eat", "tea", "tan"], Output: [["eat", "tea"], ["tan"]]

```
def groupAnagrams(strs):
    groups = {}
    for s in strs:
        key = ''.join(sorted(s))
        if key not in groups:
            groups[key] = []
        groups[key].append(s)
    return list(groups.values())
```

Explanation:

- Anagrams have same sorted characters.
- Sort each string as a key for the dictionary.
- Group strings with same key together.
- Return list of groups.
- **Edge Cases**: Empty list, single string.

---

10. Isomorphic Strings

Problem: Check if two strings are isomorphic (one-to-one character mapping).
Example: Input: s = "egg", t = "add", Output: true

```
def isIsomorphic(s, t):
    if len(s) != len(t):
        return False
    s2t = {}
    t2s = {}
    for c1, c2 in zip(s, t):
        if s2t.get(c1, c2) != c2 or t2s.get(c2, c1) != c1:
            return False
        s2t[c1] = c2
        t2s[c2] = c1
    return True
```

Explanation:

- Isomorphic strings have a one-to-one character mapping.
- Use two dictionaries to map `s` to `t` and `t` to `s` .
- Check if mappings are consistent.
- **Edge Cases**: Different lengths, no mapping.

Linked Lists

11. Reverse Linked List

Problem: Reverse a singly linked list.
Example: Input: 1->2->3, Output: 3->2->1

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def reverseList(head):
    prev = None
    current = head
    while current:
        next_node = current.next
        current.next = prev
        prev = current
        current = next_node
    return prev
```

Explanation:

- Use three pointers: `prev`, `current`, `next_node`.
- Reverse links by pointing `current.next` to `prev`.
- Move pointers forward until end.
- **Edge Cases**: Empty list, single node.

---

12. Detect Cycle in a Linked List

Problem: Check if a linked list has a cycle.
Example: Input: 1->2->3->2(cycle), Output: true

```
def hasCycle(head):
    slow = fast = head
    while fast and fast.next:
        slow = slow.next
        fast = fast.next.next
        if slow == fast:
            return True
    return False
```

Explanation:

- Use two pointers: `slow` moves one step, `fast` moves two.
- If they meet, there's a cycle.
- If `fast` reaches end, no cycle.
- **Edge Cases**: Empty list, no cycle.

---

13. Merge Two Sorted Lists

Problem: Merge two sorted linked lists into one sorted list.
Example: Input: 1->2->4, 1->3->4, Output: 1->1->2->3->4->4

```
def mergeTwoLists(list1, list2):
    dummy = ListNode(0)
    current = dummy
    while list1 and list2:
        if list1.val <= list2.val:
            current.next = list1
            list1 = list1.next
        else:
            current.next = list2
            list2 = list2.next
        current = current.next
    current.next = list1 if list1 else list2
    return dummy.next
```

Explanation:

- Use a dummy node to build merged list.
- Compare nodes from both lists, pick smaller, move forward.
- Attach remaining nodes.
- **Edge Cases**: Empty lists.

---

14. Remove N-th Node From End

Problem: Remove the n-th node from the end of a linked list.
Example: Input: 1->2->3->4->5, n = 2, Output: 1->2->3->5

```
def removeNthFromEnd(head, n):
    dummy = ListNode(0, head)
    slow = fast = dummy
    for _ in range(n + 1):
        fast = fast.next
    while fast:
        slow = slow.next
        fast = fast.next
    slow.next = slow.next.next
    return dummy.next
```

Explanation:

- Use two pointers: `fast` moves `n+1` steps ahead.
- Move both pointers until `fast` reaches end; `slow` is at node before target.
- Skip target node.
- **Edge Cases**: Remove head, single node.

Trees

15. Binary Tree Inorder Traversal

Problem: Traverse binary tree in-order (left-root-right).
Example: Input: [1,null,2,3], Output: [1,3,2]

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def inorderTraversal(root):
    result = []
    def inorder(node):
        if node:
            inorder(node.left)
            result.append(node.val)
            inorder(node.right)
    inorder(root)
    return result
```

Explanation:

- In-order: visit left, root, right.
- Use recursion to traverse tree.
- Append node values to `result`.
- **Edge Cases**: Empty tree, single node.

---

16. Level Order Traversal (BFS)

Problem: Traverse binary tree level by level.
Example: Input: [3,9,20,null,null,15,7], Output: [[3],[9,20],[15,7]]

```
from collections import deque

def levelOrder(root):
    if not root:
        return []
    result = []
    queue = deque([root])
    while queue:
        level = []
        for _ in range(len(queue)):
            node = queue.popleft()
            level.append(node.val)
            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)
        result.append(level)
    return result
```

Explanation:

- Use queue for BFS (level-by-level).
- Process all nodes at current level, add children to queue.
- Store each level's values in `result`.
- **Edge Cases**: Empty tree, single level.

---

17. Maximum Depth of Binary Tree

Problem: Find the maximum depth of a binary tree.
Example: Input: [3,9,20,null,null,15,7], Output: 3

```
def maxDepth(root):
    if not root:
        return 0
    left_depth = maxDepth(root.left)
    right_depth = maxDepth(root.right)
    return max(left_depth, right_depth) + 1
```

Explanation:

- Depth is longest path from root to leaf.
- Recursively find depths of left and right subtrees.
- Return max of both plus 1 (for root).
- **Edge Cases**: Empty tree, single node.

---

18. Lowest Common Ancestor (BST)

Problem: Find lowest common ancestor in a binary search tree.
Example: Input: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 8, Output: 6

```
def lowestCommonAncestor(root, p, q):
    while root:
        if p.val < root.val and q.val < root.val:
            root = root.left
        elif p.val > root.val and q.val > root.val:
            root = root.right
        else:
            return root
    return None
```

Explanation:

- BST: left subtree < root < right subtree.
- If both `p` and `q` are less than root, go left.
- If both greater, go right. Else, root is LCA.
- **Edge Cases**: Nodes not in tree, same node.

Stacks & Queues

19. Valid Parentheses

Problem: Check if a string of parentheses is valid.
Example: Input: s = "()", Output: true

```
def isValid(s):
    stack = []
    pairs = {')': '(', '}': '{', ']': '['}
    for char in s:
        if char in '({[':
            stack.append(char)
        elif char in ')}]':
            if not stack or stack.pop() != pairs[char]:
                return False
    return len(stack) == 0
```

Explanation:

- Use stack to track opening brackets.
- For closing brackets, check if it matches top of stack.
- Stack should be empty at end.
- **Edge Cases**: Empty string, unmatched brackets.

---

20. Min Stack

Problem: Design a stack that supports push, pop, top, and getMin in O(1).
Example: push(-2), push(0), push(-3), getMin() -> -3

```
class MinStack:
    def __init__(self):
        self.stack = []
        self.min_stack = []

    def push(self, val):
        self.stack.append(val)
        if not self.min_stack or val <= self.min_stack[-1]:
            self.min_stack.append(val)

    def pop(self):
        if self.stack.pop() == self.min_stack[-1]:
            self.min_stack.pop()

    def top(self):
        return self.stack[-1]

    def getMin(self):
        return self.min_stack[-1]
```

Explanation:

- Use two stacks: one for values, one for minimums.
- Push: Add to `stack`; if `val` <= current min, add to `min_stack`.
- Pop: Remove from `stack`; if it's min, remove from `min_stack`.
- **Edge Cases**: Empty stack.

---

21. Evaluate Reverse Polish Notation

Problem: Evaluate RPN expression.
Example: Input: ["2", "1", "+", "3", "*"], Output: 9

```
def evalRPN(tokens):
    stack = []
    for token in tokens:
        if token in "+-*/":
            b = stack.pop()
            a = stack.pop()
            if token == "+":
                stack.append(a + b)
            elif token == "-":
                stack.append(a - b)
            elif token == "*":
                stack.append(a * b)
            elif token == "/":
                stack.append(int(a / b))
        else:
            stack.append(int(token))
    return stack[0]
```

Explanation:

- Use stack to process RPN.
- Numbers: push to stack.
- Operators: pop two numbers, compute, push result.
- **Edge Cases**: Single number, division by zero.

Greedy & Intervals

22. Merge Intervals

Problem: Merge overlapping intervals.
Example: Input: [[1,3],[2,6],[8,10],[15,18]], Output: [[1,6],[8,10],[15,18]]

```
def merge(intervals):
    intervals.sort()
    result = [intervals[0]]
    for current in intervals[1:]:
        if current[0] <= result[-1][1]:
            result[-1][1] = max(result[-1][1], current[1])
        else:
            result.append(current)
    return result
```

Explanation:

- Sort intervals by start time.
- Compare current interval with last merged: if overlap, update end; else, add new interval.
- **Edge Cases**: Empty list, single interval.

---

23. Insert Interval

Problem: Insert new interval and merge if needed.
Example: Input: intervals = [[1,3],[6,9]], newInterval = [2,5], Output: [[1,5],[6,9]]

```
def insert(intervals, newInterval):
    result = []
    i = 0
    n = len(intervals)
    while i < n and intervals[i][1] < newInterval[0]:
        result.append(intervals[i])
        i += 1
    while i < n and intervals[i][0] <= newInterval[1]:
        newInterval[0] = min(newInterval[0], intervals[i][0])
        newInterval[1] = max(newInterval[1], intervals[i][1])
        i += 1
    result.append(newInterval)
    while i < n:
        result.append(intervals[i])
        i += 1
    return result
```

Explanation:

- Add intervals before `newInterval`.
- Merge overlapping intervals with `newInterval`.
- Add remaining intervals.
- **Edge Cases**: Empty intervals, newInterval at start/end.

---

24. Non-overlapping Intervals

Problem: Remove minimum intervals to make rest non-overlapping.
Example: Input: [[1,2],[2,3],[3,4],[1,3]], Output: 1

```
def eraseOverlapIntervals(intervals):
    if not intervals:
        return 0
    intervals.sort()
    count = 0
    end = intervals[0][1]
    for i in range(1, len(intervals)):
        if intervals[i][0] < end:
            count += 1
            end = min(end, intervals[i][1])
        else:
            end = intervals[i][1]
    return count
```

Explanation:

- Sort intervals by start time.
- Track earliest end time. If next interval overlaps, remove one with later end.
- Count removals.
- **Edge Cases**: Empty list, no overlaps.

Binary Search

25. Binary Search

Problem: Find target in sorted array.
Example: Input: nums = [1,3,5,6], target = 5, Output: 2

```
def search(nums, target):
    left, right = 0, len(nums) - 1
    while left <= right:
        mid = (left + right) // 2
        if nums[mid] == target:
            return mid
        elif nums[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1
```

Explanation:

- Split sorted array in half, check middle.
- If target equals middle, return index.
- If target > middle, search right half; else, left half.
- **Edge Cases**: Empty array, target not found.

---

26. Search in Rotated Sorted Array

Problem: Find target in rotated sorted array.
Example: Input: nums = [4,5,6,7,0,1,2], target = 0, Output: 4

```
def search(nums, target):
    left, right = 0, len(nums) - 1
    while left <= right:
        mid = (left + right) // 2
        if nums[mid] == target:
            return mid
        if nums[left] <= nums[mid]:
            if nums[left] <= target < nums[mid]:
                right = mid - 1
            else:
                left = mid + 1
        else:
            if nums[mid] < target <= nums[right]:
                left = mid + 1
            else:
                right = mid - 1
    return -1
```

Explanation:

- Rotated array has sorted halves.
- Check if left or right half is sorted, then decide where target lies.
- Adjust `left` or `right` based on comparison.
- **Edge Cases**: Single element, no rotation.

Recursion/Backtrack

27. Subsets

Problem: Find all possible subsets of an array.
Example: Input: nums = [1,2,3], Output: [[],[1],[2],[3],[1,2],[1,3],[2,3],[1,2,3]]

```
def subsets(nums):
    result = [[]]
    for num in nums:
        result += [subset + [num] for subset in result]
    return result
```

Explanation:

- Start with empty subset `[[]]`.
- For each number, add it to all existing subsets.
- Build new subsets iteratively.
- **Edge Cases**: Empty array.

---

28. Combination Sum

Problem: Find combinations that sum to target.
Example: Input: candidates = [2,3,6,7], target = 7, Output: [[2,2,3],[7]]

```
def combinationSum(candidates, target):
    result = []
    def backtrack(remain, current, start):
        if remain == 0:
            result.append(current[:])
            return
        if remain < 0:
            return
        for i in range(start, len(candidates)):
            current.append(candidates[i])
            backtrack(remain - candidates[i], current, i)
            current.pop()
    backtrack(target, [], 0)
    return result
```

Explanation:

- Use backtracking to try all combinations.
- If sum equals target, add to result.
- If sum exceeds target, stop.
- **Edge Cases**: No solution, empty candidates.

---

29. Permutations

Problem: Find all permutations of an array.
Example: Input: nums = [1,2,3], Output: [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]

```
def permute(nums):
    result = []
    def backtrack(nums, current):
        if len(current) == len(nums):
            result.append(current[:])
            return
        for num in nums:
            if num not in current:
                current.append(num)
                backtrack(nums, current)
                current.pop()
    backtrack(nums, [])
    return result
```

Explanation:

- Use backtracking to build permutations.

- Add each unused number to current permutation.
- When length matches, add to result.
- **Edge Cases**: Empty array, single element.

Graphs

30. Number of Islands

Problem: Count number of islands (connected 1s) in a grid.
Example: Input: [["1","1","0"],["1","1","0"],["0","0","0"]], Output: 1

```
def numIslands(grid):
    if not grid:
        return 0
    rows, cols = len(grid), len(grid[0])
    count = 0
    def dfs(i, j):
        if i < 0 or i >= rows or j < 0 or j >= cols or grid[i][j] != "1":
            return
        grid[i][j] = "0"
        dfs(i+1, j)
        dfs(i-1, j)
        dfs(i, j+1)
        dfs(i, j-1)
    for i in range(rows):
        for j in range(cols):
            if grid[i][j] == "1":
                dfs(i, j)
                count += 1
    return count
```

Explanation:

- Use DFS to mark connected 1s as visited (0).
- Count each new island (1) found.
- Explore all four directions.
- **Edge Cases**: Empty grid, no islands.

---

31. Clone Graph

Problem: Clone an undirected graph.
Example: Input: node 1->2,2->3,3->1, Output: cloned graph

```
class Node:
    def __init__(self, val=0, neighbors=None):
        self.val = val
        self.neighbors = neighbors if neighbors is not None else []

def cloneGraph(node):
    if not node:
        return None
    clones = {}
    def dfs(node):
        if node in clones:
            return clones[node]
        clone = Node(node.val)
        clones[node] = clone
        for neighbor in node.neighbors:
            clone.neighbors.append(dfs(neighbor))
        return clone
    return dfs(node)
```

Explanation:

- Use DFS to clone each node and its neighbors.

- Store clones in dictionary to avoid duplicates.
- Recursively clone neighbors.
- **Edge Cases**: Empty graph, single node.

Dynamic Programming

32. Climbing Stairs

Problem: Find number of ways to climb n stairs (1 or 2 steps).
Example: Input: n = 3, Output: 3

```
def climbStairs(n):
    if n <= 2:
        return n
    a, b = 1, 2
    for _ in range(3, n + 1):
        a, b = b, a + b
    return b
```

Explanation:

- Each step can be 1 or 2 stairs.
- Number of ways for n = ways for (n-1) + ways for (n-2).
- Use two variables to track previous steps.
- **Edge Cases**: n = 1, n = 2.

---

33. House Robber

Problem: Find maximum amount robbing non-adjacent houses.
Example: Input: [1,2,3,1], Output: 4 (1+3)

```
def rob(nums):
    if not nums:
        return 0
    if len(nums) <= 2:
        return max(nums)
    a, b = nums[0], max(nums[0], nums[1])
    for i in range(2, len(nums)):
        a, b = b, max(b, a + nums[i])
    return b
```

Explanation:

- Can't rob adjacent houses.
- Track max money up to current house and previous house.
- Choose max of (current + non-adjacent) or (previous).
- **Edge Cases**: Empty array, single house.

---

34. Longest Palindromic Substring

Problem: Find longest palindromic substring.
Example: Input: s = "babad", Output: "bab"

```
def longestPalindrome(s):
    n = len(s)
    start = max_length = 0
    for i in range(n):
        len1 = expandAroundCenter(s, i, i)
        len2 = expandAroundCenter(s, i, i + 1)
        length = max(len1, len2)
        if length > max_length:
            max_length = length
            start = i - (length - 1) // 2
    return s[start:start + max_length]

def expandAroundCenter(s, left, right):
    while left >= 0 and right < len(s) and s[left] == s[right]:
        left -= 1
        right += 1
    return right - left - 1
```

Explanation:

- Check palindromes by expanding around each center.
- Try odd (center at i) and even (center between i, i+1) lengths.
- Track longest palindrome's start and length.
- **Edge Cases**: Single character, empty string.

Final Tips

- Practice these on CodeSignal's "Arcade" or "Interview Practice" sections.
- Time yourself (30-40 minutes per problem).
- Test edge cases (empty inputs, single elements).
- Explain logic aloud to mimic interview.
- Leverage your AI Chatbot and AI Finance projects to explain problem-solving.

Somanath, these 34 solutions cover DevRev's likely questions! Your Python skills are perfect for this. Practice hard, and you're set for a 90%+ success rate! Good luck!